

Walkabout: An Asynchronous Messaging Architecture for Mobile Devices



Adam Hudson

A thesis submitted in fulfillment
of the requirements for the degree of
Doctor of Philosophy

School of Information Technologies
The University of Sydney

March 2008

© 2008 - Adam Hudson

Author
Adam Hudson

Thesis supervisor
Bob Kummerfeld

Abstract

Modern mobile devices are prolific producers and consumers of digital data, and wireless networking capabilities enable them to transfer their data over the Internet while moving. Applications running on these devices may perform transfers to upload data for backup or distribution, or to download new content on demand. Unfortunately, the limited connectivity that mobile devices experience can make these transfers slow and impractical as the amount of data increases.

This thesis argues that asynchronous messaging supported by local proxies can improve the transfer capabilities of mobile devices, making it practical for them to participate in large Internet transfers. The design of the Walkabout architecture follows this approach: proxies form store-and-forward overlay networks to deliver messages asynchronously across the Internet on behalf of devices. A mobile device uploads a message to a local proxy at rapid speed, and the overlay delivers it to one or more destination devices, caching the message until each one is able to retrieve it from a local proxy. A device is able to partially upload or download a message whenever it has network connectivity, and can resume this transfer at any proxy if interrupted through disconnection.

Simulation results show that Walkabout provides better throughput for mobile devices than is possible under existing methods, for a range of movement patterns. Upload and end-to-end transfer speeds are always high when the device sending the message is mobile. In the basic Walkabout model, a message sent to a mobile device that is repeatedly moving between a small selection of connection points experiences high download and end-to-end transfer speeds, but these speeds fall as the number of connection points grows. Pre-emptive message delivery extensions improve this situation, making fast end-to-end transfers and device downloads possible under any pattern of movement.

This thesis describes the design and evaluation of Walkabout, with both practical implementation and extensive simulation under real-world scenarios.

Acknowledgements

My name is on the front cover of this thesis, but I would not have been able to produce it without the academic, emotional and financial support of a large group of people.

Firstly, I'd like to thank my supervisor Bob Kummerfeld. He has been guiding my research since 1999, starting with my third year project, through honours and finally my doctorate. His depth of knowledge has been invaluable, and he has always been available to sit down and talk through the problems that I have faced. It has been a joy to work with him, and I doubt that I could have achieved the level of success that I have experienced at university without his help.

Aaron Quigley also provided excellent help in his early role as my associate supervisor. The long discussions that we shared helped me to see my research problem from different angles while the ideas were still forming, and his enthusiasm for research was inspirational. Judy Kay has never officially been a supervisor, but she has always provided valuable support in her role as co-leader of the SIT/CHAI research group. I'm also grateful for the pep talks that I've received from Alan Fekete and Björn Landfeldt, which gave me motivation and new directions to explore at key times.

The "G61B" crew of Trent Apted, Mark Assad, Michael Avery, David Carmichael, Derek Corbett, Dan Cutting, Daren Ler, William Niu, David Storey and David Symonds (as well as honorary members Charles Prabhakar and Tara McIntosh) have all been great friends, research collaborators and travel companions over the past four-and-a-half years. It has been a pleasure to share the day-to-day highs and lows of being PhD student with them, and I hope to continue working with them in the future.

Of this group, there are several people who I need to thank in particular for the help that they have provided. Mark Assad for patiently putting up with my stupid questions about programming, networking and nearly everything else. He would provide the correct answer every single time, quickly and concisely, and sometimes even answer questions before I asked them. Dan Cutting for his constant discussion, encouragement and rivalry, even when he was living on the other side of the planet. He won the race, but still helped me to the very end. David "Furry" Symonds, for working with Dan to produce the indispensable ROMNeT simulation framework, for providing all sorts of simulation and mathematical advice, and for

offering valuable input during the final few weeks. Andrew Lum for the insights on what it takes to successfully complete a PhD thesis. These guys have always been there to provide advice when I needed it, even when they were dealing with the stresses of their own research.

When the time came to produce the final copy of this document, numerous people were there to provide valuable proofreading services. I'd like to thank Peter Aylett, Derek Corbett, Dan Cutting, Alan Fekete, Sarah Hudson, Judy Kay, Tara McIntosh, David Storey, David Symonds and Rainer Wasinger for the time that they contributed to help me out.

I'm grateful for the financial support that the Smart Internet Technology CRC has provided to my studies. They have also given me the opportunity to meet interesting and like-minded students from across the country and the world.

I'd also like to thank the entire academic, admin, workshop and programming staff of the School of IT for the various support that they've provided over the years. Particular thanks go to James Curran, for access to his group's servers when I was running out of computing power.

A big thank you goes to my parents, grandparents and brothers for their constant support and encouragement during the long time that I've spent at university. They haven't seen much of me over the past few years, but have still always been there for me when I needed them. Now that I'm finished, they will finally get their son/grandson/brother back!

The greatest thanks of all must go to my wonderful wife Sarah. I know that the stresses of producing a thesis have made life difficult at times, yet Sarah has always been there to provide every kind of support imaginable, without ever uttering a single complaint and often without receiving any show of gratitude. She enabled me to block out the rest of the world when I needed to so that I could focus solely on my research. There is absolutely no way that I could have made it through this degree in one piece without her. Adding to this, she is also a major contributor to this document, responsible for producing most of the diagrams.

I know that was a lot of people, but I owe a great debt to every one of them. There is no way that I can actually capture the full extent of my gratitude, but this hopefully gives some idea. Thank you all!

Publications

Some of the material and concepts in this thesis have appeared in published conference papers and technical reports.

Conferences

HUDSON, A. & KUMMERFELD, B. (2006). Walkabout: Asynchronous messaging support for mobile devices. In *Proceedings of the 3rd International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2006)*, 215–222, London, UK.

HUDSON, A., KUMMERFELD, B. & QUIGLEY, A. (2004). A file migration architecture for pervasive systems. In *Adjunct Proceedings of the 6th International Conference on Ubiquitous Computing (UbiComp 2004)*, Nottingham, UK.

Technical Reports

HUDSON, A. & KUMMERFELD, B. (2006). Walkabout: An asynchronous Internet messaging architecture tailored to mobile devices. Tech. Rep. TR-579, School of IT, The University of Sydney.

Contents

List of Figures	xv
List of Tables	xix
A note on data units	xxi
Acronyms	xxiii
1 Introduction	1
1.1 Motivation	1
1.2 Local proxies	2
1.3 Walkabout	4
1.4 Contributions	6
1.5 Thesis outline	7
2 Applications	9
2.1 Devices	9
2.2 Upload to a server	11
2.3 Download from a server	12
2.4 Transfers between devices	14
2.5 Discussion	14
3 Background	17
3.1 Service discovery	18
3.2 Peer-to-peer overlay networks	20
3.2.1 Structured overlays	20
3.2.2 File transfer overlays	22
3.2.3 BitTorrent	23
3.3 Internet-scale mobile device communication	25
3.3.1 Registry-based	26

3.3.2	Redirection-based	29
3.3.3	Storage-based	32
3.3.4	Opportunistic networking	34
3.4	Proxies	36
3.4.1	Infostations	37
3.5	Mobile device movement prediction	38
3.6	Summary	39
4	Core system design	41
4.1	Overview	41
4.1.1	Features	43
4.2	Components	44
4.2.1	Distributed object location and routing service	44
4.2.2	Service discovery	44
4.2.3	Client	45
4.2.4	Proxy	45
4.2.5	Message tracker and the overlay	46
4.3	Message delivery process	46
4.3.1	Client connection	47
4.3.2	Message generation	48
4.3.3	Producer upload	49
4.3.4	Overlay transfer	50
4.3.5	Consumer download	54
4.3.6	Completion	55
4.4	Cache management	56
4.5	Short messages	56
4.6	Fault tolerance	57
4.7	Security	58
4.8	Deployment	60
4.9	Prototype implementation	62
4.9.1	Client library API	66
4.9.2	Client application interface	67
4.10	Summary	68

5	Core system evaluation	69
5.1	Experimental setup	69
5.1.1	Test network	69
5.1.2	Evaluation criteria	71
5.2	Exploration of base parameters	72
5.2.1	Request window size	73
5.2.2	Piece size	77
5.2.3	Recommendations	80
5.3	Comparison to existing techniques	82
5.3.1	Fixed producer and fixed consumer	83
5.3.2	Fixed producer and multiple fixed consumers	85
5.3.3	Mobile producer and fixed consumer	86
5.3.4	Fixed producer to mobile consumer	90
5.4	Summary	101
 6	 Extended system design	 103
6.1	Client manager	104
6.1.1	Manager creation	105
6.1.2	Client connection and disconnection	105
6.1.3	Prediction algorithm	106
6.1.4	Message distribution policy	107
6.1.5	Message completion	111
6.1.6	Additional functionality	111
6.2	Identity tracker	111
6.2.1	Tracker creation	112
6.2.2	Device and rule registration	113
6.2.3	Delivering messages	113
6.2.4	Groups	114
6.2.5	User agents	115
6.3	Simple device agents	115
6.4	Summary	117
 7	 Client manager evaluation	 119
7.1	Experimental setup	119
7.1.1	Test network	119

7.1.2	Evaluation criteria	120
7.2	Predict-on-disconnection	122
7.2.1	Perfect prediction	122
7.2.2	Imperfect prediction	125
7.2.3	Markov prediction	127
7.3	Supernode	132
7.4	Summary	138
8	Conclusions and future directions	141
8.1	Contributions	141
8.2	Future directions	143
8.2.1	Message scheduling	143
8.2.2	Support for streaming media	144
8.2.3	Media transcoding	145
8.2.4	Client manager modules	146
8.2.5	Other possibilities	147
	References	149

List of Figures

1.1	Average advertised broadband downlink speed by country, October 2007	3
1.2	File delivery using an overlay network.	5
2.1	A camera uses Walkabout to upload photos to a home server.	12
2.2	A PlayStation Portable using “remote play” to connect to a PlayStation 3. . . .	15
3.1	Different approaches to service discovery.	19
3.2	A BitTorrent download.	24
3.3	Registry-based communications.	28
3.4	Redirection-based communications.	29
3.5	Storage-based communications.	33
3.6	An example of opportunistic networking.	35
4.1	A Walkabout overlay network.	42
4.2	A producer continues uploading a message as it moves between proxies.	49
4.3	The use of pieces versus blocks.	51
4.4	Piece and block flow in a complex delivery scenario.	55
4.5	A summary of the number of access points recorded by WiGLE.	61
4.6	Coverage maps of the metropolitan WiFi networks in Mountain View, California and Philadelphia, Pennsylvania.	61
4.7	The main window of the backup utility, showing the current progress.	64
4.8	The host entry and selection dialogue of the backup utility.	64
5.1	The test network.	70
5.2	The contents of a single sub-network.	70
5.3	Impact of RWS on the average effective transfer speed.	74

5.4	Impact of RWS and the number of consumers on the average effective transfer speed.	76
5.5	Impact of piece size on average data overheads.	78
5.6	Impact of message size on the average effective transfer speed.	78
5.7	Breakdown of the contribution each notification type makes to overheads.	79
5.8	Impact of the number of consumers on the average effective transfer speed.	81
5.9	Impact of the number of consumers on the source proxy stress.	81
5.10	Impact of the number of consumers on the average effective transfer speed between a fixed producer and multiple fixed consumers.	85
5.11	Impact of the number of consumers on average data overheads between a fixed producer and multiple fixed consumers.	87
5.12	Impact of connection time on the average effective transfer speed between a mobile producer and a fixed consumer.	87
5.13	Impact of connection time on the average effective transfer speed between a mobile producer and a fixed consumer under Walkabout, when network links are asymmetrical.	89
5.14	Impact of disconnection time on the average effective transfer speed between a fixed producer and a mobile consumer.	91
5.15	Impact of disconnection time on the average consumer download speed for a mobile consumer.	92
5.16	Impact of disconnection time on average data overheads between a fixed producer and a mobile consumer.	93
5.17	Impact of disconnection time on the average consumer download speed between a fixed producer and a mobile consumer, when network links are asymmetrical.	94
5.18	Impact of disconnection time on the average effective transfer speed between a fixed producer and a mobile consumer, when network links are asymmetrical.	95
5.19	Impact of disconnection time on average data overheads between a fixed producer and a mobile consumer, when network links are asymmetrical.	96
5.20	The main campus of Dartmouth College.	98
5.21	Path lengths derived from the Dartmouth college movement traces.	99
5.22	Impact of disconnection time on the average consumer download speed between a fixed producer and a mobile consumer following real-world movement patterns.	100

5.23	Impact of disconnection time on average data overheads between a fixed producer and a mobile consumer following real-world movement patterns.	100
6.1	The layered client manager components.	105
6.2	The predict-on-disconnection message distribution policy in action.	108
6.3	The supernode message distribution policy in action.	110
6.4	An identity tracker and its identity group.	112
6.5	A simple device uploading to its backup server.	116
7.1	Impact of perfect prediction on the average consumer download speed.	123
7.2	Impact of perfect prediction on average data overheads.	123
7.3	Impact of varying prediction accuracy on the average consumer download speed. Line intersections correspond to recorded data points.	126
7.4	Impact of varying prediction accuracy on average data overheads.	126
7.5	Crossover points between imperfect prediction and unmanaged transfers.	127
7.6	Accuracy of the Markov predictor for each client across all traces with more than 100 transitions.	129
7.7	Accuracy of the Markov predictor for each of the top fifty client traces.	129
7.8	Impact of Markov prediction on the average consumer download speed, for real-world movement traces.	130
7.9	Impact of Markov prediction on average data overheads, for real-world move- ment traces.	130
7.10	Average percentage of message downloads per-proxy, for real-world traces.	131
7.11	The supernode test network.	133
7.12	Impact of the supernode MDP on the average consumer download speed, for 500KB/s downlinks.	134
7.13	Impact of the supernode MDP on the average data overheads, for 500KB/s downlinks.	134
7.14	Impact of the supernode MDP on the average consumer download speed, for 1MB/s downlinks.	137

List of Tables

1.1	Network speeds for commonly available technologies.	3
2.1	Examples of powerful portable devices.	10
3.1	Structured overlay APIs.	21
3.2	Summary of approaches to mobile device communication.	27
4.1	Message header fields.	48
5.1	Optimal RWS for future experiments.	77
5.2	Walkabout transfer performance between two fixed devices.	83
5.3	Experimental parameters for comparison to existing techniques, Chapter 5 . . .	84
5.4	Average producer upload speed.	88
7.1	Experimental parameters for Chapter 7	121
7.2	Experimental results for a 200MB transfer using perfect prediction, with 10 second connection and 30 minute disconnection periods.	124
7.3	Supporting variables for Equation 7.1.	135

A note on data units

In general, this thesis will use the following basic units for data measurement:

- 1 byte (B) = 8 bits
- 1 kilobyte (KB) = 1024 bytes
- 1 megabyte (MB) = 1024 kilobytes
- 1 gigabyte (GB) = 1024 megabytes

Accordingly, all experimental data rates are measured in kilobytes per second (KB/s) or megabytes per second (MB/s).

Somewhat confusingly, internet service providers and equipment manufacturers often measure data rates with units that use bits (rather than bytes) and multiples of 1000 (rather than 1024). Commonly used units include the following:

- 1 kilobit per second (Kbps) = 1000 bits per second
- 1 megabit per second (Mbps) = 1000 kilobits per second
- 1 gigabit per second (Gbps) = 1000 megabits per second

Rather than converting these values, they will be left in the original units where appropriate, but the reader is advised to bear in mind that 1Mbps is approximately equal to 122KB/s.

Acronyms

Acronym	Expanded term
ADSL	Asymmetric Digital Subscriber Line
ADU	Application Data Unit
AES	Advanced Encryption Standard
API	Application Programming Interface
CAN	Content Addressable Network
CFS	Cooperative File System
CoDoNS	Cooperative Domain Name System
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DNS	Domain Name System
DNS-SD	Domain Name System Service Discovery
DOLR	Distributed Object Location and Routing
DVB	Digital Video Broadcasting
FTP	File Transfer Protocol
FTTH	Fibre to the Home
GPS	Global Positioning System
GUI	Graphical User Interface
HIP	Host Identity Protocol
HSPA	High-Speed Packet Access
i3	Internet Indirection Infrastructure
IP	Internet Protocol
ISP	Internet Service Provider
JEDI	Java Event-based Distributed Infrastructure
JMS	Java Message Service
MAC	Media Access Control
MDP	Message Distribution Policy

Acronym	Expanded term
MMP	Mobile Motion Prediction
MMS	Multimedia Messaging Service
NFS	Networked File System
PKI	Public Key Infrastructure
PC	Personal Computer
PS3	PlayStation 3
PSP	PlayStation Portable
PVR	Personal Video Recorder
ROAM	Robust Overlay Architecture for Mobility
RWS	Request Window Size
SHA	Secure Hash Algorithm
SLP	Service Location Protocol
SMS	Short Message Service
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
TTL	Time-To-Live
UIP	Unmanaged Internet Protocol
UMPC	Ultra-Mobile Personal Computer
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
USB	Universal Serial Bus
WiGLE	Wireless Geographic Logging Engine

Chapter 1

Introduction

1.1 Motivation

Modern mobile electronic devices are becoming increasingly adept at handling digital multimedia in all its forms. Fast processors, gigabytes worth of storage and a rich array of input and output technologies all support their ability to create, store and render multimedia. Because they are portable, mobile devices give people the freedom to experience and create content any time and place that they want.

Connectivity is also an important part of their functionality. Mobile devices regularly interact with fixed devices and other mobile devices to exchange content, whether it is to store locally created content for safekeeping, to share it with a group of interested people, or to acquire new content on demand. Direct local links over USB or a local area network can support these transfers at high data rates, but in today's connected world it should also be possible to perform the same operations on a worldwide scale across the Internet.

Fortunately, the networking capabilities of mobile devices are also increasing, with high-speed wireless networking interfaces from the 802.11 family (commonly referred to as WiFi) and cellular networking becoming commonplace. WiFi access points are also widespread, deployed by individuals and organisations alike. These technologies enable mobile devices to access the Internet directly, and exchange digital content with any other device in the world without being tethered to a fixed network. However, the nature of these devices means that even though this form of communication is usually possible, it is rarely *practical*.

One reason for this is that a mobile device is not always reachable. It may be out of the range of a suitable network, or switched off to conserve power. This can create difficulties if another device wishes to contact it to initiate a direct transfer. This situation is compounded

if the initiating device is also mobile and has to contend with the same connectivity issues. Cellular networking that is “always on” offers one way to overcome this problem, but data costs can be prohibitive, speed is limited, coverage is not absolute, and relatively few devices beyond mobile phones have the required hardware.

Another problem is that the media content involved can be quite large, ranging in size from tens of kilobytes for small images, to several megabytes for typical photos and music files, all the way up to gigabytes for high definition video. In many cases, transferring this content across the Internet can require minutes or possibly even hours of constant connectivity. For a mobile device with intermittent connectivity, the actual time taken between the transfer initiation and completion can be longer still. If a person wishes to complete a large transfer in the shortest possible time, they may find that they need to modify or restrict their movements to guarantee device connectivity for the duration of the transfer.

It is clear that the unreliable connectivity associated with mobility can make it difficult to transfer large amounts of data across the Internet. This thesis aims to make these large transfers practical, by improving the networking abilities of mobile devices. **In this context, a transfer is defined as practical if:**

- **it succeeds irrespective of the connection patterns of the devices involved;**
- **the speed is comparable to what would be observed between fixed hosts;**
- **the user of a mobile device can achieve this speed without needing to alter their movement patterns.**

1.2 Local proxies

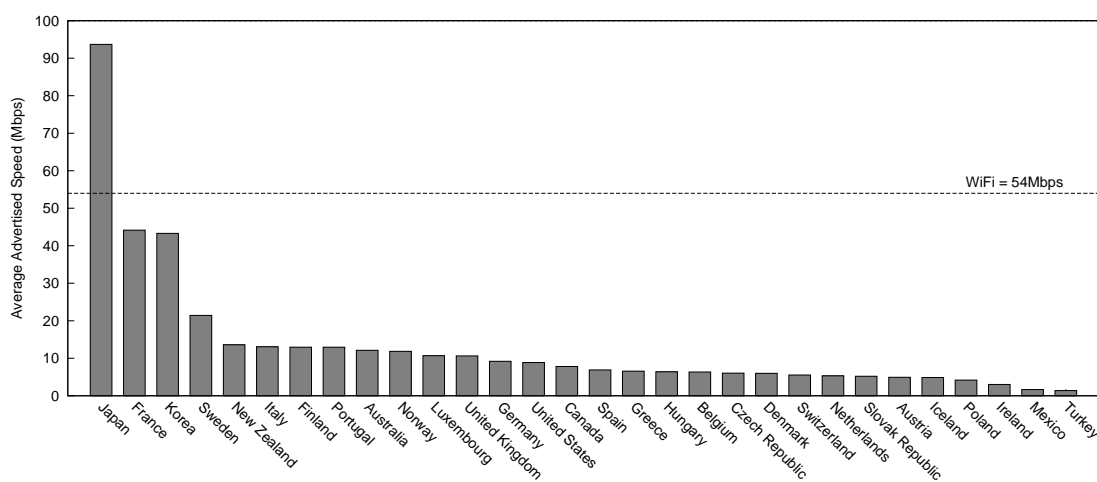
In general, mobile devices can achieve higher transfer rates when communicating across a local wireless network than across the Internet. This is partly due to the difference in the speeds that the different technologies offer. Table 1.1 presents the maximum speeds of some consumer-level networking technologies that are commonly available in 2007. It shows that the maximum speed of local 802.11g WiFi exceeds that of the Internet access methods of ADSL2+ or High Speed Packet Access (HSPA), particularly on the uplink. Figure 1.1 indicates that WiFi is also faster than the average consumer broadband downlink speed available in the majority of countries. The average connection speeds available in Japan, France and Korea approach or exceed those of WiFi, and emerging technologies like Fibre to the Home (FTTH) promise to improve these speeds further to 1Gbps (Takachi *et al.*, 2006), but these values do not

Table 1.1: Network speeds for commonly available technologies.

Technology	Type	Max Downlink (Mbps)	Max Uplink (Mbps)
802.11g WiFi	Local network	54	54
ADSL2+	Fixed Internet	24	3.5
HSPA	Cellular Internet	14.4 [†]	1.45 [‡]

[†]Highest commercially available speed (Telstra, 2007)

[‡]Highest commercially available speed (hsupa.com, 2007)

**Figure 1.1:** Average advertised broadband downlink speed by country, October 2007 (OECD, 2007).

necessarily improve actual transfer speeds beyond what local wireless can offer. An Internet transfer passes through many links along its path, so a slow or congested link at any point will restrict the overall throughput. Local transfers pass through a relatively small number of links, so there is more chance of them attaining their maximum speed, and advances in technology continue to offer ever-higher maximum speeds. For example, 802.11n WiFi is due for release in 2008 and will provide local speeds of up to 300Mbps, with non-standard “pre-n” equipment available already (Apple Computer, Inc., 2007), and short-range wireless speeds are estimated to reach 15Gbps within the next three years (GEDC, 2007; Press Esc, 2007). The other factor is that it may be some time before technologies like FTTH or high-speed cellular access become widespread, because it is difficult and expensive for telecommunications companies to deploy them. Meanwhile, it is relatively easy to purchase and install new WiFi equipment as faster speeds become available.

Local proxies provide a way to exploit the disparity between local network and Internet speeds. A proxy is defined as an agent that mediates communication between a device and the wider network. If proxies reside within the same local networks as mobile devices, they

can take advantage of the speed disparity to make large Internet transfers practical, by acting as local storage points to buffer *asynchronous messaging* between the devices.

In the asynchronous messaging paradigm, the communication endpoints never contact each other directly, but exchange messages via some intermediate infrastructure instead. Email, SMS and MMS are common existing asynchronous communication methods. This store-and-forward approach is perfectly suited to transfers between mobile devices, as it serves to minimise the effect of fluctuating network connectivity. The sender is able to upload data messages at its convenience, without any knowledge of or dependency on recipient availability. The infrastructure stores the messages and delivers them as soon as the recipient is available. Asynchronous messaging decouples the endpoints, so that they can communicate without concern for each other's connection patterns. In the extreme case, devices are able to communicate without any common connection time at all.

The presence of local proxies can provide great benefits for asynchronous messaging. When sending messages, a device can offload them at local speed to the proxy, which caches them and forwards them across the Internet at the slower speed. The Internet transfer is therefore able to continue even if the device disconnects. When delivering messages, the local proxy accepts messages on behalf of the receiving device. If this device is connected as the messages arrive, the proxy forwards them immediately and there is no advantage. However, if the device is not connected, the proxy caches the messages, then forwards them when the device next connects. This allows the device to “catch up” on the download at local speeds upon reconnection, as if it had never disconnected. Effectively, the proxies prevent the Internet from being a communication bottleneck, so that the devices are able to make more efficient use of their connection periods. As a result, mobile devices should be able to carry out large transfers even with only sporadic network connectivity.

These properties lead to the central hypothesis of this research: **that asynchronous messaging supported by local proxies makes it practical for mobile devices to transfer large data objects across the Internet.**

1.3 Walkabout

To test this hypothesis, an asynchronous messaging architecture called Walkabout is proposed, which takes advantage of local proxies to support Internet data transfers. Walkabout proxies are located within network segments across the Internet and *client* devices connect to them locally, typically over WiFi. *Producer* clients upload messages to their nearest proxy,

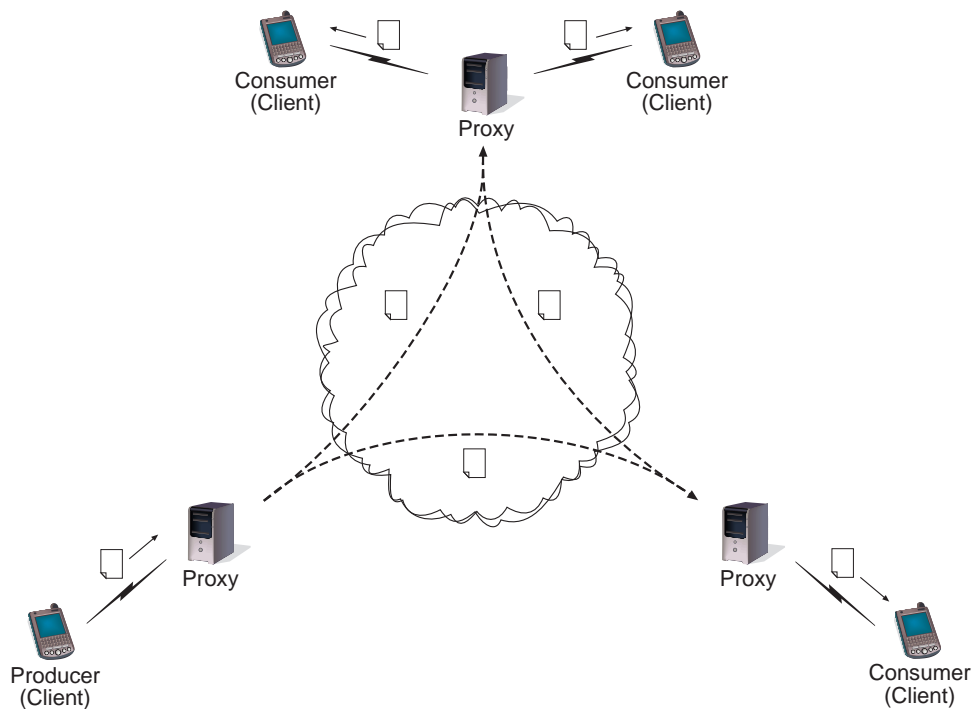


Figure 1.2: File delivery using an overlay network.

addressed to one or more *consumer* clients, and the proxy creates a peer-to-peer Internet overlay with other proxies to cooperatively deliver each message, without the need for data storage on or coordination by a central authority. Figure 1.2 shows these components, arranged to form an overlay for the delivery of a file to three consumers. The proxies buffer message uploads and downloads, so that clients can transfer at local speeds whenever possible. If disconnection interrupts a message transfer, the client can resume it at any proxy upon reconnection.

This design provides the necessary support to build mobile applications based upon asynchronous messaging. Messages can range in size from small inter-application messages, up to large media files in the order of gigabytes. While multimedia transfer is a motivation for this research, Walkabout is not restricted to this domain and can support any other type of content, including documents, games or scientific data. The only condition is that messages must be discrete, with the size defined before the transfer begins. Examples of applications that this messaging can enable are explored in depth in Chapter 2.

Walkabout's message-based approach is ideally suited to the delivery of personal data. Content created on a device can be uploaded quickly to a variety of different targets. Valuable data, such as holiday photos and videos, can be uploaded to a host for permanent storage. Content with limited appeal, such as video messages, can be sent directly to the

devices of a group of interested people. If the content is of general interest and wider distribution is sought, the producer can deliver it directly to a content sharing site like Flickr (Yahoo!, 2007b) or YouTube (Google, 2007e). Walkabout's directed messaging can also provide a speed boost when downloading personal content, such as that drawn from a personal library, or copy-protected content purchased directly from an online store. This improvement would traditionally only be available when downloading popular data, which has been cached locally in response to previous downloads or replicated in anticipation of future demand.

This thesis presents the full design for Walkabout and investigates its performance through simulation under a variety of device movement scenarios, including patterns based upon real-world traces. This investigation shows significant improvements in upload, download and end-to-end transfer speeds when compared to alternative transfer techniques. These results show that Walkabout achieves its aim of providing mobile devices with a practical means to transfer large data items across the Internet, irrespective of mobility patterns.

1.4 Contributions

This thesis makes three major contributions:

- C1:** A review of the literature associated with an architecture that facilitates Internet data transfers involving mobile devices.
- C2:** The design of a distributed network architecture that enables asynchronous Internet message delivery to one or more devices, with particular focus on mobile senders and receivers. Local network proxies enable high upload speeds for all devices, and high download speeds for devices with repetitive movement patterns, resulting in end-to-end transfer speeds that are largely unaffected by mobility. When an uploading device is highly mobile, the architecture can improve end-to-end transfer speeds beyond what is possible between fixed systems.
- C3:** The design of a pre-emptive message delivery service on top of the basic architecture. The service uses different movement prediction and message delivery policies to deliver high download speeds for devices with any movement pattern.

1.5 Thesis outline

Chapter 2 - Applications: Presents further motivation for Walkabout, through an exploration of some applications that it will enable and existing devices that can run them.

Chapter 3 - Background: Gives an overview of the background material that is necessary to understand the design decisions behind Walkabout and describes the related work. This forms Contribution **C1**.

Chapter 4 - Core system design: Details the design of the core Walkabout architecture, including its components, operation and API.

Chapter 5 - Core system evaluation: Evaluates the performance of Walkabout through simulation. Together with Chapter 4, this forms Contribution **C2**.

Chapter 6 - Extended system design: Details the design of additional features to make Walkabout applicable to a wider range of situations. These features are a pre-emptive message delivery service, greater recipient control over message delivery and network support for simple devices.

Chapter 7 - Client manager evaluation: Evaluates the performance of two pre-emptive delivery mechanisms through simulation. Together with Chapter 6, this forms Contribution **C3**.

Chapter 8 - Conclusions and future directions: Concludes this thesis and presents directions for future research.

Chapter 2

Applications

This chapter presents some of the devices that Walkabout is designed to support and a range of potential applications that they could run. These Internet-based applications are a major motivation for this work and fall in to three distinct categories: uploads from mobile devices to fixed servers, downloads to mobile devices from fixed servers, and transfers between mobile devices. The chapter concludes by examining some existing applications that are similar, but that operate without the benefit of local proxies.










2.1 Devices

A wide range of mobile devices are available on the market today, featuring fast processors, gigabytes of local storage, and many different functions. Each of the recently released devices featured in Table 2.1 has a variety of features, which may include a large, high resolution screen, image and audio capture hardware, speaker/headphone output, a keyboard or touch-screen input. These features enable them to create, store and render a wide variety of different content, most notably photos, videos, music, documents and games. Importantly, even though each of the devices in the table has a different purpose, they all feature built-in WiFi that enables them to send and receive this content over both local networks and the Internet.

As explained in Chapter 1, Walkabout aims to harness the wireless local networking ability of these mobile devices, to enable them to transfer data at local speeds wherever possible. This allows for cheap and quick Internet transfers, without significant impact upon a person's mobility. The following sections illustrate just some of the applications that become possible when there is a rapid form of asynchronous messaging available to a mobile device. They demonstrate the need for this type of messaging and provide supporting scenarios for the simulations in later chapters.

2.1. Devices

Table 2.1: Examples of powerful portable devices.

			
Device:	Apple MacBook	Sony Vaio UMPC	HP iPAQ
Description:	Notebook computer	Ultra-mobile PC	Pocket PC
Network:	WiFi, Bluetooth	WiFi, Bluetooth, cellular	WiFi, Bluetooth
Images:	Create, view	Create, view	View
Video:	Create, view	Create, view	View
Audio:	Record, listen	Record, listen	Record, listen
Documents:	Create, view	Create, view	Create, view
Plays games:	Yes	Yes	Yes
			
Device:	Kodak EasyShare-One	Nokia N93i	Nokia N800
Description:	Camera	Phone	Internet tablet
Network:	WiFi (Bluetooth model also available)	WiFi, Bluetooth, cellular	WiFi, Bluetooth
Images:	Create, view	Create, view	Create, view
Video:	Create, view	Create, view	Create, view
Audio:	Record, listen	Record, listen	Record, listen
Documents:	N/A	Create, view	Create, view
Plays games:	No	Yes	Yes
			
Device:	Microsoft Zune	Sony PlayStation Portable (PSP)	iLiad iRex
Description:	Digital media player	Entertainment device	e-book reader
Network:	WiFi	WiFi	WiFi
Images:	View	View	View
Video:	View	View	N/A
Audio:	Listen	Listen	Listen
Documents:	View	View	Create, view
Plays games:	No	Yes	No

2.2 Upload to a server

There is a great need for mobile devices to be able to upload their data across the Internet to a safe location on a regular basis. The data that these devices capture can be irreplaceable, yet their portable nature commonly places them in situations where they can be lost, stolen or damaged. The ability to regularly upload files to a remote location would protect them from loss. While mobile devices can have large amounts of onboard or card-based storage, it is unfortunately very easy to create or download far more content than a device can hold. As storage fills up, users reach the point where they have to become more selective about what new content they create, or choose to sacrifice existing content to free up space, both of which can be very undesirable. If the device is constantly backing up files across the Internet, then the user can reclaim space as needed by deleting the local copies. The location to upload data to would need to be a reliable server, which could be managed personally, by an organisation or provided by a company for a fee.

The rise in popularity of social networking sites that are driven by user-created content provides further motivation for this type of immediate upload. Sites such as YouTube (Google, 2007e) (which presents videos), Flickr (Yahoo!, 2007b) (photos) and Blogger (Google, 2007a) (text and multimedia) inspire people to publish content, often captured on a mobile device, to the world. The ability to publish directly from the device would increase the accessibility of these sites even further, so that they can share media at any time, from practically any location. It would also provide an extremely powerful means for ordinary people to report news as it happens.

With these motivations in mind, the following applications demonstrate how Walkabout improves the upload abilities of mobile devices.

Photo backup: *An Australian on holiday in Europe carries his EasyShare-One camera and 512MB memory card wherever he goes. He takes hundreds of photos every day, each one between 3 and 5MB in size. The camera automatically uploads each photo to a machine in his home within minutes of capture (see Figure 2.1). When the memory card starts running low on space, the camera automatically deletes photos that are safely backed up.*

Walkabout enables the traveller to take as many photos as he wants with only a single memory card, and his photos remain safe even if something happens to the camera. The camera performs the backup regardless of the route he follows over the course of a day.

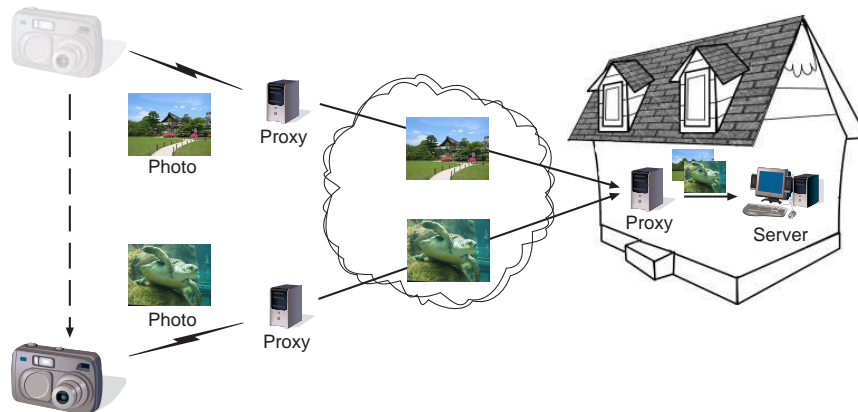


Figure 2.1: A camera uses Walkabout to upload photos to a home server.

Direct publication: *A gang of thieves attempt a bank heist, but a courageous security guard stops them all single-handedly. A bystander captures the entire event on her N93i phone and publishes the 20MB video directly to YouTube via a WiFi hotspot. The initial upload only takes seconds, the video is available on the web within minutes, and people all round the world are watching it long before any news teams arrive on the scene.*

Walkabout enables the woman to publish the content to the web quickly from her current location, with only minimal connection time required.

2.3 Download from a server

Although modern mobile devices can hold gigabytes of data, this may only be a subset of what a person owns. This means that a mobile device may be missing content that becomes necessary as they move around over time. Even though a person may anticipate what content they expect to need and load it on to their device from a larger library while still at home, there is always the chance that they will need something different later on. This could possibly be because they want to show it to someone else, because it has only just become available, or simply because they changed their mind. Walkabout messaging opens up possibilities for users to acquire content from a remote server, and load it directly on to their devices while mobile. The data transfers will take place opportunistically in the background as they continue to carry out their normal activities. Requests could either be made explicitly through an interface, or the content could be sent automatically by way of a schedule or user agent running on the remote server.

The following applications are examples of how Walkabout improves the abilities of mobile devices to download data from the Internet.

Online purchasing: *A jogger is listening to music on her Zune, but finds that a song she heard on the radio that morning is stuck in her head. She pauses momentarily within an area of WiFi coverage, where she connects the Zune to an online store, purchases the song, and adds it as a virtual item to her current playlist. She returns to jogging while the device downloads the 5MB file in the background. The download is complete within a couple of minutes, then the song starts playing as the next item in her playlist.*

Walkabout enables the jogger to request high quality media content from a vendor on demand, and to download the file opportunistically while still moving, rather than having to wait in one location until the download completes.

Media retrieval: *While in the car on the way to his parents' house, a man remembers that he promised his mother that he would bring last year's Christmas photos with him. He pulls over to the side of the road, browses a locally cached catalogue of photos on his Nokia N800 Internet tablet, and requests the 100MB album of photos from his home media server. The download starts instantly and continues throughout the course of the drive, as he passes through areas of public WiFi coverage along the way. By the time he sits down to afternoon tea with his family, the photos are available to show to them.*

Walkabout enables the man to download a large number of items from a personal library, that he did not have the opportunity to retrieve while he was at home. He can download them while continuing his trip, which is a faster option than returning home or downloading them when he reaches his destination.

Automatic media delivery: *A student's personal video recorder (PVR) is set to record her favourite show each week, and transmit it automatically to her Vaio UMPC. If she is at home, it simply does so over local wireless. However, one week she is overseas at a conference, so the PVR sends it over the Internet instead. The Vaio downloads the 200MB show in the background as she moves between presentations at the conference, so that when she returns to the hotel that evening, it is ready to watch.*

Walkabout once more enables a large download without requiring any modifications to mobility patterns. In this instance, the transfer is initiated automatically, rather than explicitly requested by the user.

2.4 Transfers between devices

Transferring content directly between mobile devices over the Internet can be difficult, as it usually requires both the sender and receiver to be connected simultaneously. Uploading via an intermediate server can be an effective way to remove this restriction, but has other problems. Media sharing sites place size and topic restrictions on what can be posted (for example, YouTube videos must not exceed 10 minutes nor 100MB), and may be too public when the content is only intended for one person or a small group. A dedicated storage server in the home or office is perhaps a better intermediate option, but users do not always have access to one. There are also issues of efficiency to consider, as uploading then downloading will tend to take longer and use more bandwidth than transferring directly. It is therefore desirable to enable communications between mobile devices without the need for an intermediate server.

The asynchronous messaging support that Walkabout offers allows mobile devices to exchange data directly, without needing simultaneous connectivity. Both uploads and downloads can take place while the devices are moving and messages can easily reach multiple destinations. SMS and MMS on mobile phones are existing examples of this technique, but the ability to transfer larger files opens up the potential for even more powerful rich messaging applications. Existing applications that enable multimedia messaging between fixed devices, such as the Keep-in-Touch family messaging system (Assad *et al.*, 2005), could easily incorporate Walkabout to extend their reach to mobile devices. The following example shows how this might work:

Video messages: *While walking home from a school, a pair of teenage girls decide that they want to have a party, so they sit down in a park to record a video invite on a MacBook. The final message is 20MB in size, and the MacBook uploads it automatically while they continue walking home, for direct delivery to the phones, notebooks and portable media players of their friends.*

Walkabout enables the girls to send a large media message directly from one mobile device to a diverse group of other devices. The message will reach each of these devices, regardless of the mobility patterns of the sender or the recipients.

2.5 Discussion

Many of the mobile applications presented in this chapter do currently exist, but they are limited by their underlying networking technology. Phone-based applications commonly

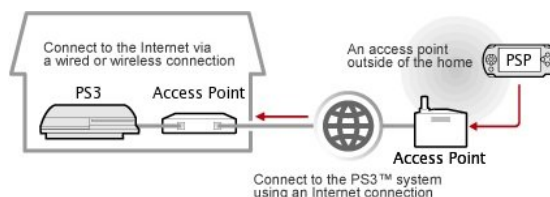


Figure 2.2: A PlayStation Portable using “remote play” to connect to a PlayStation 3 (Sony Computer Entertainment, 2007).

make use of the cellular multimedia messaging service (MMS) (Open Mobile Alliance, 2005), to exchange images, audio, video and text with both fixed and mobile endpoints, albeit at a limited size and quality. More generally, email allows any kind of content to be sent as an attachment over either cellular or local network connections to the Internet. The issues of speed and cost discussed in Chapter 1 therefore apply to any transfers that use these methods.

YouTube (Google, 2007f), Flickr (Yahoo!, 2007a) and Blogger (Google, 2007b) all provide facilities for content to be uploaded from mobile devices, by MMS or email. The ZoneTag application also manages mobile photo uploads to Flickr, with additional location tags drawn automatically from GPS or cell data (Yahoo!, 2007d). The EasyShare-One camera is able to use any WiFi access point to upload photos directly to Kodak’s online subscription-based gallery (Kodak, 2005). These upload services might be suitable for the occasional upload, but not for large scale transfers such as large videos or full camera backups. However, with these services already in place, it would be trivial for these companies to provide Walkabout support. An upload application running on the consumer’s device could send messages to a server-side application, which acts as a gateway between Walkabout and the existing storage service. The customer could use WiFi to upload these messages in bursts while moving around, but without incurring high cellular transfer costs.

Most mobile phone service providers now offer music and video that can be streamed directly to the phone. Some also offer digital music that can be purchased, downloaded and stored locally, an example being Sprint’s music store (Sprint Nextel, 2007). Owners of Sony’s PSP can use its “remote play” capabilities to connect to a PlayStation 3 (PS3) home entertainment console across the Internet and stream images, music or video over WiFi (Sony Computer Entertainment, 2007). Figure 2.2 is drawn from Sony’s web site, and shows how simple it can be to establish this link. While streaming is a popular option for these companies (partially because it restricts the ability of end-users to redistribute the content), it requires a constant connection, which either limits transfers to cellular links, or cuts down on mobility when WiFi is used. If the content is large or the device’s owner is moving around rapidly, downloading

it over Walkabout would be a better option. A cellular connection could still provide backup connectivity, for the situation when a download is urgent but there are no WiFi access points within immediate range. Alternatively, a user could use the connection to place a request for a large file from a server, then receive it as a regular Walkabout transfer over WiFi.

The applications in this chapter illustrate the potential benefits of enabling mobile devices to send and receive large amounts of data. While major companies have already expressed interest in providing similar applications, their approaches are impractical when large content is involved. This is due to slow transfer times, high data costs and the limitations they impose on mobility. The design for Walkabout proposed in this thesis improves the capabilities of mobile devices, by making it practical for people to use data transfer applications in a broad range of situations, such as those illustrated by the scenarios in this chapter. These scenarios form the basis of the evaluations in the later chapters of this thesis.

Chapter 3

Background

The main focus of this thesis is an architecture that facilitates Internet data transfers involving mobile devices. This chapter presents a review of the research that informs the design of the architecture, and covers several key areas:

- *Service discovery*: How mobile devices automatically discover and connect to network services, both locally and across the Internet.
- *Peer-to-peer overlay networks*: The use of peer-to-peer overlay networks to locate mobile clients and data objects, and to transfer large amounts of data across the Internet.
- *Mobile device location and communication*: Different ways that a host can make contact with a mobile device on demand anywhere on the Internet, and continue to communicate even as it moves.
- *Proxies*: Ways that proxy hosts can improve the Internet transfer abilities of mobile devices.
- *Movement prediction*: How to predict the future movement patterns of a mobile devices, and how this knowledge can help to improve data delivery.

The following sections explore existing research in each of these areas, to identify the strengths and weaknesses of the different approaches, and how the principles that they present relate to Walkabout. They describe all of the background material that is necessary to understand the architectural design covered in the remaining chapters of this thesis.

3.1 Service discovery

Hosts in a local-area network can provide each other with a variety of different services, such as Internet access, file repositories and printers. For an application running on a device to make use of these services, it must be aware that they exist, and know how to contact them. The simplest approach is to manually configure applications with details of the services they will use, but this requires careful administration to keep details current if service availability changes. A well-defined service discovery protocol provides a more flexible alternative, by introducing a common mechanism for applications to automatically discover which hosts can provide the services that meet their needs. This becomes particularly important when devices are mobile, as they need to be able to discover the local services any time they connect to an unfamiliar network.

There are many different local area service discovery protocols in existence today. Each protocol has a unique approach, but they still share fundamental similarities. They all allow the application to send out a query for services that match given criteria, and to receive responses that detail which hosts can provide them. A protocol may achieve this by following a centralised directory-based approach, a distributed approach, or a combination of the two:

- *Directory-based*: Providers advertise a service description to a dedicated directory, which applications query directly to discover service information. A host needs to know a directory's location to communicate with it, so this must be pre-configured, provided during DHCP registration, or discovered initially using a distributed method as below. The directory infrastructure may be distributed across a number of hosts to improve reliability and scalability. Examples: Jini (Sun Microsystems, 2001; Waldo, 1999), Salutation (The Salutation Consortium, 1999) and the Service Discovery Service (SDS) which is part of the Ninja project (Czerwinski *et al.*, 1999)
- *Distributed*: Applications send out a service query to all local hosts, typically via multicast, and receive responses from any host that can provide a matching service. A service provider may also advertise services directly to the network, and other hosts cache these details for future reference. Examples: Bluetooth Service Discovery Protocol (SDP) (Bluetooth SIG, 2003), MOCA (Beck *et al.*, 1999) and the Simple Service Discovery Protocol (SSDP) within Universal Plug and Play (UPnP) (UPnP Forum, 2003).
- *Both*: Some protocols are able to operate in either mode, giving them options for both scalability and redundancy. Examples: Service Location Protocol (SLP) (Guttman *et al.*, 1999; Guttman, 1999), Bonjour (Apple Computer, Inc., 2006), JXTA (Gong, 2002).

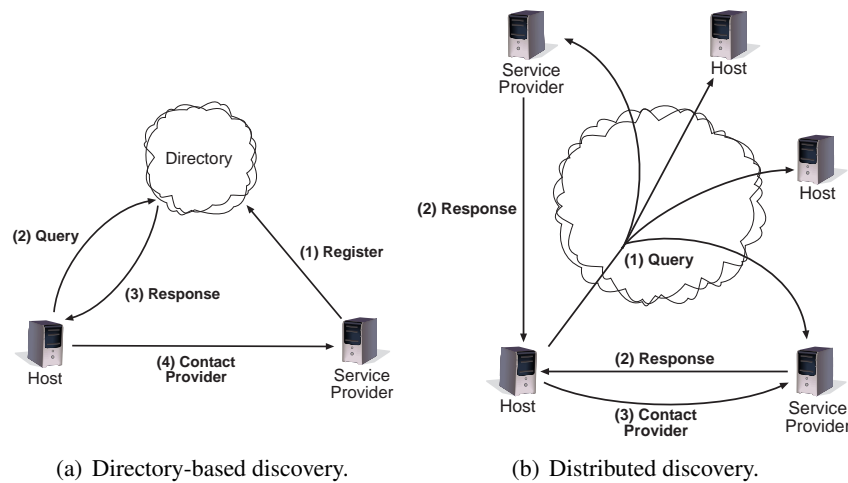


Figure 3.1: Different approaches to service discovery.

Figure 3.1 shows the difference between directory-based and distributed service discovery protocols. The zero-configuration approach of distributed discovery makes it suitable for small or ad hoc networks, but the amount of network traffic generated increases with the number of hosts and services. Thus larger networks are better served by directory-based discovery, where the cost of creating and maintaining a directory is offset by improvements in efficiency and scalability.

Protocols that make use of a directory may also be able to advertise services beyond the local network, although not all are designed to do so. Apple’s Bonjour is based upon Domain Name System Service Discovery (DNS-SD) (Cheshire & Krochmal, 2006a), and is an example of a protocol that can work on both a local and global scale. In a local network, discovery typically operates in a distributed fashion over multicast DNS (Cheshire & Krochmal, 2006b), but the providers can also choose to register their services with a DNS server that supports dynamic updates (Vixie *et al.*, 1997). If this DNS server is globally addressable, then devices outside the local network can also discover these services, if they are configured to search that domain.

Individual protocols have other features, such as enhanced security, an expressive query language or management of the actual connection between application and service, and these are covered in depth in other surveys (Rakotonirainy & Groves, 2002; Zhu *et al.*, 2005). For mobile devices, though, the ability to quickly and easily locate services in any network is the key property they require of a service discovery protocol, and any of the options mentioned here would be appropriate.

3.2 Peer-to-peer overlay networks

An overlay network presents a virtual network topology that is independent of the underlying physical structure. The abstraction allows for network node addressing, neighbour selection and routing rules that suit the application creating the overlay. They are also known as peer-to-peer networks because the structure is organised by the very hosts that use the service, all of which typically perform equal roles as peers (as opposed to a client-server arrangement). Peer-to-peer communications offer a number of reliability and performance improvements over client-server communications, and this section details these improvements. There are many different types of overlays, but this review focuses on those most relevant to Walkabout: structured overlays, which are used to locate devices and content; and file transfer overlays, which are used to transfer files between proxies. Particular attention is focused on the BitTorrent file transfer overlay protocol, which Walkabout's transfer overlays are based upon.

3.2.1 Structured overlays

Structured overlay networks connect nodes together according to a well-defined algorithm, to enable the efficient location of data objects even when there are a large number of nodes. A host typically connects to the overlay by contacting an existing member, and establishes itself as a node with a random, unique identity. This identity dictates its position within in a shared coordinate space, and thus which other peers are its neighbours. To route a message to a specific address, a node starts by delivering it to one of its neighbours. The message follows a route through nodes that are progressively “closer” to the destination (based upon some metric particular to the overlay) until it arrives. The delivery is scalable, as it typically takes no more than $O(\log n)$ steps for a network consisting of n nodes, and reliable, as multiple routes exist to the same destination. Examples of structured overlay systems include Chord (Stoica *et al.*, 2001), Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao *et al.*, 2001), CAN (Ratnasamy *et al.*, 2001) and Kademlia (Maymounkov & Mazieres, 2002). A survey paper by El-Ansary & Haridi (2005) summarises and compares these systems (and more) in detail.

Distributed Hash Tables (DHTs) and Distributed Object Location and Routing (DOLR) are two major services that can be built on top of these routing overlays. The difference between these services lies in the APIs defined by Dabek *et al.* (2003), which Table 3.1 summarises. In both services, a node that holds a data object assigns a unique *key* to it, which is drawn from the same address space as the overlay nodes. This is usually obtained by taking a cryptographic hash of some property of the object (such as the file name). The node in the overlay with the closest address to this key is known as its *root*.

Table 3.1: Structured overlay APIs.

Overlay	Function	Description
DHT	<code>put(key, data)</code>	Store <i>data</i> at the root node for <i>key</i> .
	<code>remove(key)</code>	Delete the data associated with <i>key</i> .
	<code>value = get(key)</code>	Retrieve the data associated with <i>key</i> .
DOLR	<code>publish(key)</code>	Announce that the data object identified by <i>key</i> is available at the publishing node.
	<code>unpublish(key)</code>	Delete all mappings from <i>key</i> to this node.
	<code>sendToObj(msg, key, [n])</code>	Deliver <i>msg</i> to the closest node that has published <i>key</i> . May optionally request to send to the closest <i>n</i> nodes.

As the name implies, DHTs provide similar key/value pair storage functionality to that of a traditional hash table data structure, but in a distributed fashion. A host stores a data object at the root node identified by the object’s key, and any host can subsequently retrieve the object using the same key.

DOLR provides a way to route messages to one or more hosts that hold a replica of a data object. When a host holds an instance of a given data object, it routes a publication towards the root node for the object. Nodes along the routing path also take note of this publication. More than one node may publish that they have the same data object. When another node wishes to contact a host that has this object, it routes a message towards the root node. The first publication encountered along the path is considered to be the closest, and so the node responsible for that publication receives the message.

Because the peers that use structured overlays also form the routing substrate, these systems are designed to run on networks that consist mainly of fixed hosts. They do possess failure detection and recovery mechanisms, but rapid changes in overlay membership can lead to routing delays, inconsistencies or failure (Li *et al.*, 2004; Rhea *et al.*, 2004). The regular movement and disconnection experienced by mobile devices can therefore limit their ability to participate and access these valuable services. One solution to this issue is to incorporate specialised recovery algorithms into the overlay (Hsiao & King, 2005; Rhea *et al.*, 2004). Alternatively, only those hosts that are likely to remain stable join the overlay, and then act as service nodes (Brampton *et al.*, 2006) or proxies (Zhao *et al.*, 2004) that enable less stable mobile nodes to access the overlay services without actually joining.

As will become apparent through the remainder of this chapter, structured peer-to-peer overlays can perform many different roles in a diverse range of applications. They

can enable network hosts to locate other hosts (Ramasubramanian & Sirer, 2004) or data objects (Cohen, 2005), and can form the basis for complex applications like file systems (Kubiatowicz *et al.*, 2000), distributed web caches (Iyer *et al.*, 2002) and messaging architectures (Castro *et al.*, 2002).

3.2.2 File transfer overlays

Peer-to-peer file transfer overlays provide a decentralised way for hosts to retrieve files on demand. Hosts download files from each other, rather than a central server, exploiting the often under-utilised upload capacity of individual hosts. Overlay formation is ad hoc, and there is no correlation between network topology and data location as there is in structured overlay networks. Only one node provides a file initially, but as more peers download it, they become additional sources for future downloads as long as they remain members of the overlay. The decentralised file publication provided by these overlays offer reliability and performance advantages over traditional centralised methods.

There are different ways to build the overlay and locate content. One approach, favoured by the Gnutella (Babenhauserheide, 2004) and FastTrack (Liang *et al.*, 2006) protocols, is for all hosts to form a single, vast overlay network. Individual hosts offer to share a number of files from their local disk, and locate content they are interested in by passing requests through the overlay. In the original Gnutella protocol, requests were given a time-to-live that decremented with each overlay hop, then simply flooded through a node's peers to a limited depth. Nodes with matching files would return their results, and the requesting node would then select the files to download. FastTrack and later versions of Gnutella refine this, so that some nodes become supernodes that index the content available at their peers and form a second tier within the overlay. A node directs its queries to a supernode, which forwards them on to its supernode peers to locate content more efficiently than if the node flooded the entire network.

Alternatively, BitTorrent (Cohen, 2003) sees hosts form individual overlays, each dedicated to transferring a single file (or collection thereof). Hosts initially find each other through a central coordinator. The next section describes the operation of BitTorrent in detail.

Once a node is a member of an overlay and has located peers that have the file, it can start downloading data. A common approach is to retrieve the file in pieces, which has several advantages. It allows nodes to maximise download speeds by retrieving different pieces of the same file from multiple peers in parallel and in arbitrary order. A node can also offer whatever pieces it has, even if it is still in the process of downloading and does not yet have the complete

file. If a file transfer is interrupted for any reason, a host can resume by reconnecting to the overlay and downloading the pieces that it still requires.

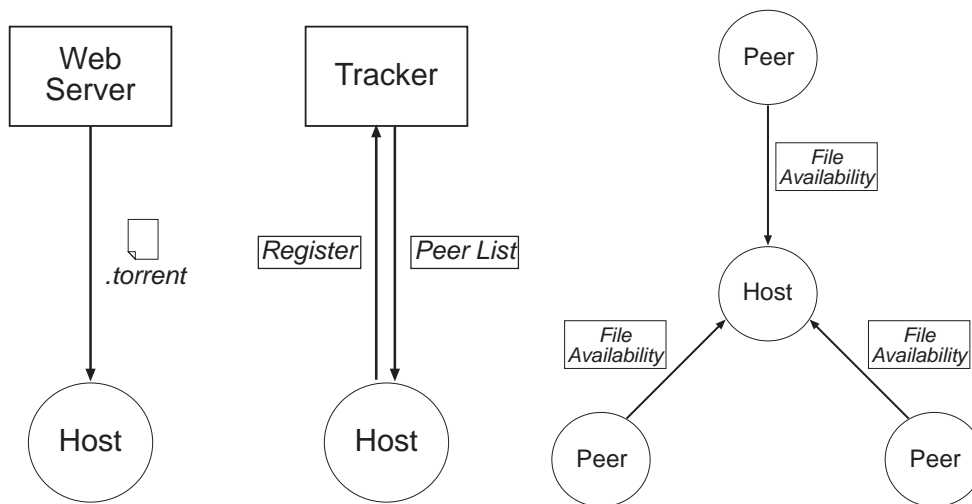
The traditional way to make a file available to a large audience is to publish it on a single server, or possibly mirror it on several servers, and for interested hosts to download it from there. If a file is popular, this can lead to high bandwidth costs for the publisher and low individual download speeds for the end user. If these servers fail for any reason, the file is no longer available. Peer-to-peer overlays improve upon the server-based approach, because availability is not dependent on a single host. The combined upload speed of all overlay members helps to provide fast downloads, while simultaneously reducing the stress on the original source. The file remains available as long as there is at least one overlay member.

Overlay networks are an extremely popular method of file distribution, with BitTorrent alone reportedly contributing to 18% of home broadband traffic as of 2006 (Ellis, 2006). They make it possible for anybody to publish content on the Internet without requiring access to a high performance server. At a bare minimum, a host needs to upload a file once in its entirety, and then the other overlay members can share the cost of distribution. A large part of their popularity is also that overlays enable people to distribute material that no service would be willing to host, such as illegally “pirated” music and video.

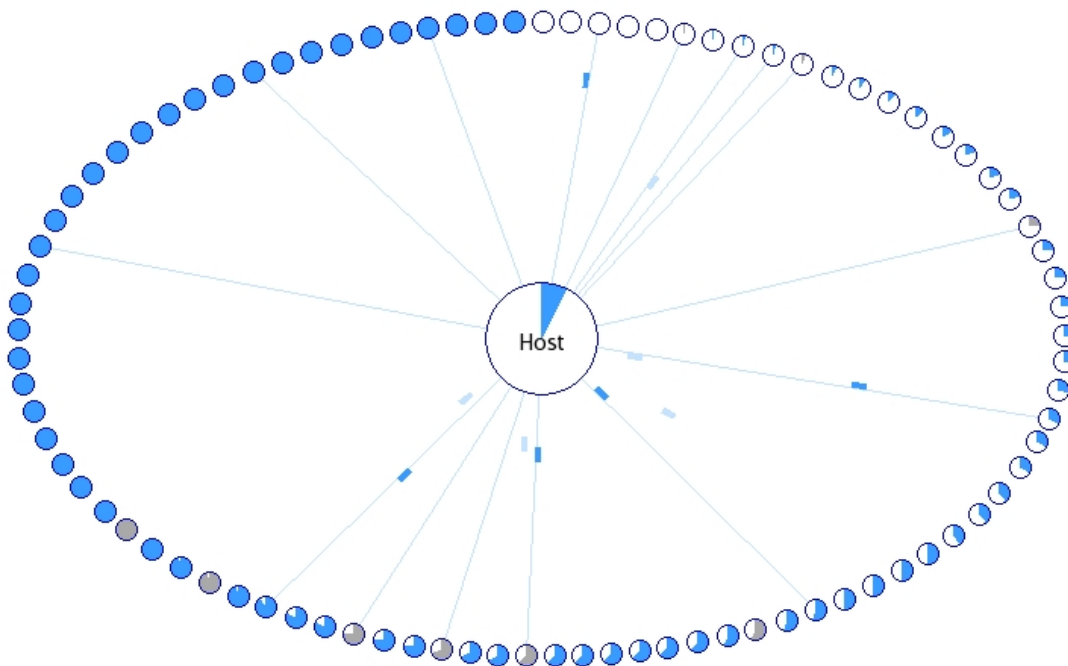
3.2.3 BitTorrent

BitTorrent (Cohen, 2003) is a protocol where overlay networks known as *torrents* are formed dynamically by hosts with mutual interest in a set of one or more files. These overlays are simple to build, but can share files effectively with either a single peer or multiple ones, making them the ideal basis for the lightweight core of Walkabout. Therefore this section explores how torrents are created and maintained.

A new torrent requires a tracker and a *.torrent* file. A *tracker* is a program that runs on a web server and coordinates the overlay, by tracking the contact details of existing peers and supplying these details to new members as they join. It also records general statistics about peer download progress. The demands on a tracker are quite low, so one will typically be responsible for the coordination of many overlays at the same time. A *.torrent* file contains important metadata about the file being published, including its name, length, hash signatures of the pieces that the file will be divided in to, and the URL for the tracker. To create a torrent, a person generates the *.torrent* for the file to be published, distributes it (usually by storing it on a web site), then introduces their host to the tracker as the initial overlay peer.



(a) Download a .torrent file. (b) Register with the tracker and retrieve the peer list. (c) Check the file availability at peers.



(d) Download from and upload to multiple peers simultaneously. The coloured regions represent the file download progress for each host (Azureus, 2007).

Figure 3.2: A BitTorrent download.

The process of downloading a torrent is presented in Figure 3.2. To begin a download, a user retrieves the .torrent file and opens it with an application. Their host contacts the tracker specified in the file to obtain a list of available peers and registers itself as a new peer. The host then contacts each of these peers to find which pieces of the file they can offer, selects smaller sub-pieces (known as blocks) for download according to rules which aim to minimise completion time while also increasing the system-wide availability of the file, and requests several of them at once from each peer. If there are a large number of suitable peers, then they are selected at random.

When a peer downloads the final block that completes a piece, it verifies the piece's hash against the one stored in the .torrent file, then alerts each of its connected peers if validation succeeds. Blocks of this piece then become available for peers to download. An essential component of BitTorrent is the choking algorithm, which aims to improve total overlay performance by ensuring a peer will preferentially upload blocks to those peers that are also willing to upload. A host continues downloading blocks until it has the whole file, at which point it stays connected to the overlay as long as possible to continue contributing to the torrent.

The tracker must be stable if a torrent is to operate smoothly. If the tracker goes offline, existing transfers between peers can continue, but no more nodes join the overlay. Modifications to the BitTorrent protocol address this central point-of-failure by making use of a Kademlia DHT to store peer information (Cohen, 2005). Portions of the torrent file are hashed to generate a globally unique identifier, and this is used as a key to store the .torrent file and the initial host's address. Subsequent hosts use this same key to register their own address and discover overlay peers in a trackerless fashion.

BitTorrent aims to minimise the average completion time for each user that is downloading the file, but the performance is dependent upon the size of the overlay and the speed of the members' uplink connections. Studies have shown that it produces consistently high average download speeds, which actually improve as the number of overlay members increase (Izal *et al.*, 2004; Pouwelse *et al.*, 2005), although Wu & Chiueh (2006) indicate that the total end-to-end delay when distributing a file to multiple recipients is sub-optimal.

3.3 Internet-scale mobile device communication

The ubiquity of IP makes it possible for hosts to communicate with each other over networks of any size. When the hosts are fixed, communication is straightforward, as an IP address is all that one host requires to route data packets to another host across any network.

Unfortunately the initialisation and maintenance of communications can become difficult when hosts are mobile.

The first issue is that of naming. A fixed host can potentially use the IP address as both an identity and a location, but this is not practical for a mobile host. Its IP address will most likely change as it moves between networks, making it difficult for a *correspondent host* (the one that is initiating a transfer) to make contact on demand. Therefore, while a mobile host's current IP address can represent its location, its identity needs to be something different so that a correspondent host can contact it by the same means no matter where it is.

The other issue is that a mobile host may move while communications are ongoing, resulting in a possible change of IP address or loss of connectivity. This is a particular problem for *synchronous communication* methods, where data flows in real-time directly between two hosts. The transport layer traditionally uses IP addresses as link endpoints, so if either host changes its address, this can interrupt their transfer. Communications can continue during host mobility if link endpoints are bound to some other location independent identity or all data passes through an intermediate fixed host, but only when both endpoints maintain network connectivity. By contrast, changes in host addresses or connectivity do not affect *asynchronous communication* methods. Hosts send messages to each other through a store-and-forward network, where they are held until the destination host is available (which may be immediately). In the extreme case, where the endpoints never have simultaneous network access, this may be the only way for them to communicate. The best approach to handle mobility depends upon the application. Some applications, particularly those based on real-time interaction, require a constant unbroken synchronous link, but others are tolerant of the potential delays that may arise from asynchronous communications.

There are a variety of different protocols and communication methods that can improve Internet transfers involving mobile devices. The different methods can be divided into four categories: registry-based; redirection-based; storage-based; and opportunistic networking. The following sections explore a number of different systems, with particular attention to how they approach naming, mobility and disconnection. Table 3.2 summarises the properties of the main systems that follow.

3.3.1 Registry-based

In the registry-based approach, a mobile device is known by a static identity. Whenever the device's IP address changes, it updates its identity's record with a location registry service. A correspondent host wishing to initiate communications with the device queries the

Table 3.2: Summary of approaches to mobile device communication.

System	Category	Addressing	Synchronous	Asynchronous
DNS	Registry	Hierarchical name	✓	✗
CoDoNS	Registry	Flat name	✓	✗
HIP	Registry	Public key	✓	✗
Mobile IP	Redirection	IP address	✓	✗
i3	Redirection	Flat name	✓	✗
UIP	Redirection	Public key	✓	✗
Warp	Redirection	Flat name	✓	✓
Topic-based pub/sub	Redirection	Topic	✓	✓
Content-based pub/sub	Redirection	None†	✓	✓
Email	Storage	User and server	✗	✓
Distributed file system	Storage	None†	✗	✓
DHT	Storage	None†	✗	✓
Tuple spaces	Storage	None†	✗	✓
Haggle	Opportunistic	Various	✗	✓

†Other than the shared knowledge of a storage location and/or data property

registry to resolve the identity to the current location, then contacts the device directly over a link that is typically synchronous. Figure 3.3 portrays this sequence of events.

The most widely deployed registry mechanism in the modern Internet is the Domain Name System (DNS) (Mockapetris, 1997a,b), which binds a host's hierarchical domain name to its IP address. It was originally designed to handle low-frequency updates, but dynamic DNS (Vixie *et al.*, 1997) introduced the ability to update name bindings in real-time. A host keeps its record current by dynamically updating the appropriate DNS server each time its address changes. An application obtains a host's IP addresses by querying its local DNS server, which contacts other DNS servers across the Internet as necessary to resolve the name. Servers cache address records to reduce lookup latency and bandwidth usage, with records having an associated time-to-live (TTL) value. Depending on the rate of mobility, a mobile host's DNS record TTL should be set to a low or zero value, to decrease the chance that caching will lead other hosts to receive outdated IP addresses.

A number of other systems advocate the use of flat namespaces as a flexible alternative for host identification. This enables hosts to derive their name from properties other than their administrative domain, as is the case with hierarchical domain names. DNS can be manipulated to support flat namespaces, as seen by the Internet fax service that resolves phone numbers via DNS to locate remote printers (Malamud & Rose, 1993). A common modern alternative is a DHT, where the hash of any name can correspond to an IP address within an Internet-scale name resolution service.

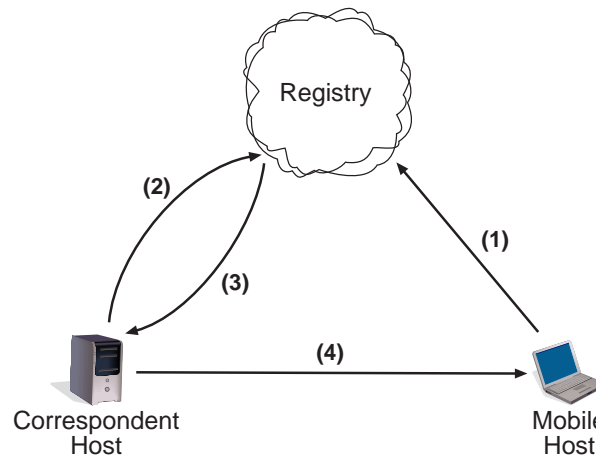


Figure 3.3: Registry-based communications: (1) the mobile host registers its current IP address; (2) the correspondent host queries the registry to (3) resolve an identifier to the mobile host’s current location; (4) the correspondent host contacts the mobile host directly.

The Cooperative Domain Name System (CoDoNS) (Ramasubramanian & Sirer, 2004) offers a service that is backwards compatible with DNS, but with improved reliability and name resolution times. IP address updates made by moving hosts are proactively propagated across the system without the need for TTL configuration. Hosts updating or retrieving address records still use hierarchical names to maintain legacy DNS compatibility, but they are hashed to a flat namespace for the purposes of the underlying DHT lookup.

The Host Identity Protocol (HIP) (Moskowitz & Nikander, 2006; Moskowitz *et al.*, 2007) introduces an additional level of indirection, that sits between the transport and network layers. A device has one or more host identifiers that are based upon public/private key pairs. They are dynamically associated with a host’s current IP address, and used by the transport layer to bind to communication endpoints. The mechanism for storing and resolving these identifiers on demand is not specifically defined, and both DNS (Nikander *et al.*, 2003) and DHTs (Eggert *et al.*, 2004) have been suggested. The underlying communications still take place over IP, but hosts dynamically update peers of any change in address, and the endpoint abstraction avoids any interruption to synchronous communications. The architecture proposed by Balakrishnan *et al.* (2004) presents a similar layered naming approach to HIP, but with further indirection that also abstracts services and data away from a particular host.

A registry-based method provides a way for a correspondent host to locate a mobile host, then carry out synchronous communications. They are simple to deploy and the direct transfers are efficient, but these methods do not provide a lot of support for mobility beyond the initial address resolution. Some approaches, such as HIP, allows for continuous

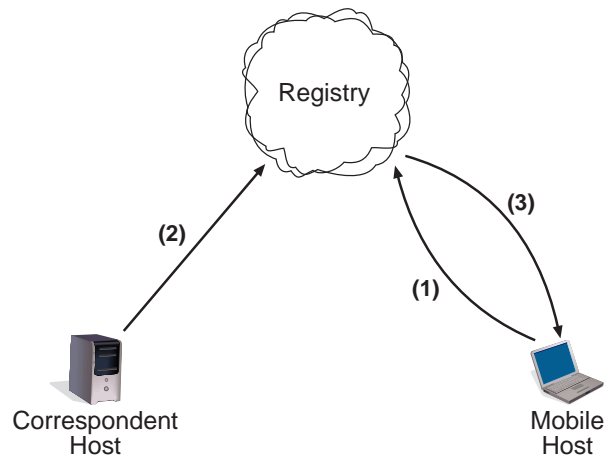


Figure 3.4: Redirection-based communications: (1) the mobile host registers its current IP address; (2) the correspondent host sends data to the registry; (3) the registry forwards the data to the mobile host.

communications if the IP address is changing. Otherwise, the introduction of mobile IP (see the next section), migration options for TCP (Snoeren & Balakrishnan, 2000) or mobile sockets (or “mockets”) (Suri *et al.*, 2005) can offer additional support.

3.3.2 Redirection-based

Redirection extends the registry concept so that the registry, rather than the correspondent host, is responsible for data delivery. The registry may be a single host acting as an agent on behalf of another, or an entire network of hosts working together to route messages. As Figure 3.4 shows, the destination host notifies a registry service any time that its IP address changes. The correspondent host sends its data via the same registry, which forwards the data on to the device’s current destination. The decoupling of the endpoints means that synchronous communications are able to continue when hosts are moving without needing to update endpoints, and that asynchronous communications are also possible if the network can store data. This also provides for a variety of flexible addressing options.

Mobile IP (version 4) (Perkins, 2002) allows a mobile host to always be reachable at the same IP address, taken from its home network, no matter where it is physically located. When the mobile host connects to a remote network, it creates a tunnel to a home agent in its home network. This home agent intercepts any IP packets addressed to the host, and forwards them over this tunnel. This hides mobility from the transport layer, so that connections can be maintained while a host is moving. The route optimisation extension (Perkins & Johnson,

2000) overcomes the inefficiencies and bottlenecks inherent in tunnelling via the home agent, by allowing the correspondent node to discover and cache mobile node IP addresses and tunnel packets to them directly.

Under the Internet Indirection Infrastructure (i3) (Stoica *et al.*, 2002), mobile hosts store *triggers* within a structured overlay network, to associate an identity with their current IP address. Public triggers have identities based on some well known property (such as a host name), while private triggers have random identities and only exist for the lifetime of a communication flow. A correspondent host starts communications by routing data (typically encapsulated IP packet payloads) to a public trigger, which redirects the data to the mobile host at its current location (if it is available). This causes the hosts to create private triggers and inform each other of the identities, then communications proceed in each direction via the nodes that host the private triggers. The mobile host updates all active triggers any time its IP address changes, so it is always reachable and all existing communications can continue uninterrupted. The Robust Overlay Architecture for Mobility (ROAM) (Zhuang *et al.*, 2003) is built upon i3, and improves the routing and handoff efficiency for mobile devices through intelligent placement of multiple triggers.

The Unmanaged Internet Protocol (UIP) (Ford, 2004) combines the self management of ad hoc networks with the scalability of IP. Hosts are identified by the hash of a public key, and the network uses these keys to route data in a peer-to-peer fashion. A host wishing to join the network starts by making contact over IP with another host that is already part of the network. Following an algorithm derived from Kademia (Maymounkov & Mazieres, 2002), it traverses the routing layer to build virtual links to peers, which it then uses to deliver data to any other host in the network.

Warp (Zhao *et al.*, 2004) is an infrastructure that enables rapid device mobility. Mobile hosts join Warp by connecting to a proxy node, which is a member of a Tapestry DOLR service. Each mobile host has a unique identity, and the proxy publishes this identity as if the host were an object on the local device. Subsequently, the proxy receives any messages routed to the identity over DOLR and forwards them to the host. When a host moves, the new proxy arranges a handover from the previous proxy through efficient alteration of the underlying Tapestry route to the identity. Warp also supports limited asynchronous communications. When a mobile host disconnects, its last proxy buffers a limited amount of data, then forwards it to the new proxy upon reconnection.

Unlike the other redirection-based methods presented in this section, the publish/subscribe (or pub/sub) messaging paradigm enables hosts to send messages to each other without

any knowledge of identities. In general, subscribers register their interest in certain events via subscription to an event broker, which may be a single server or a network thereof. A publisher generates a message and delivers it to the event broker, which forwards in a *notification* to all subscribers with matching subscriptions. This subscription may be *topic-based* and simply name a message topic, or be *content-based* and define a complex expression to filter upon the metadata and content of the message. The Java Message Service (JMS) API (Sun Microsystems, 2002) is an example of a topic-based pub/sub system, while Elvin (Segall & Arnold, 1997), Siena (Carzaniga *et al.*, 1998) and JEDI (Cugola *et al.*, 2001) are all content-based. The reader is directed to the survey paper by Eugster *et al.* (2003) for a more exhaustive discussion of pub/sub systems and their individual features.

The basic pub/sub model provides effective synchronous communications for mobile hosts. A host re-registers its subscriptions, and thus its current location, each time it reconnects to a network, but it only receives notifications if it is connected at the time of publication. However, there are several extended systems that also enable asynchronous communications, by allowing subscriptions to persist even if the subscriber is absent.

The Java Event-based Distributed Infrastructure (JEDI) (Cugola *et al.*, 2001) is a pub/sub architecture with native support for mobility. Before detaching, a mobile host can instruct its event broker to suspend its subscriptions and store any notifications that arrive in its absence. The host then reactivates its subscription and retrieves any stored notifications upon reconnection. JEDI supports multiple hierarchically arranged event brokers, so if a host reconnects to a different broker than it suspended at, notifications migrate to the new one for delivery. JMS (Sun Microsystems, 2002) also provides similar functionality through durable subscriptions, though it is an API, so the specifics of how this is achieved is dependent upon the actual implementation.

The mobility support service presented by Caporuscio *et al.* (2003) is independent of any underlying pub/sub architecture. A host transfers its subscriptions to a stationary proxy in the local network before it disconnects, and the proxy continues to collect notifications on its behalf. When the host reconnects to the pub/sub service, it also contacts its nearest proxy and retrieves any buffered notifications. If this is a different proxy to the one it left, then the new proxy contacts the old one to acquire these notifications.

Elvin supports asynchronous communications through the introduction of a proxy (Sutton *et al.*, 2001) at a central location. The proxy receives subscriptions from hosts and then registers them with the event broker. The proxy receives notifications and forwards them to the host if it is available, or otherwise holds them until it is. Even if there are multiple proxies

available within the network, the mobile host must connect back to the same one to retrieve stored notifications.

The work of Burcea *et al.* (2004) suggests ways that these systems can reduce the time spent migrating stored notifications between event brokers or proxies upon host reconnection, most notably through prefetching and logging. A prefetching event broker predicts where a host that has just disconnected will move to, and pre-emptively transfers its subscriptions and stored notifications. Alternatively, event brokers can log the identifiers of all the notifications they receive, so that when the time comes to migrate the them, it is easy to reduce the set down to only those that the new event broker does not already have.

Redirection-based communication systems provide good support for mobile devices, because they hide mobility from the sender. Some systems also enable asynchronous communication, so that transfers can even continue if the device disconnects. However, if the amount of data being transferred is large, the indirect delivery can lead to inefficiencies, and asynchronous delivery may fail if limited buffering prevents the system from storing all the data until a mobile device reconnects.

3.3.3 Storage-based

Storage-based communication solutions are asynchronous by nature, and typically deal with discrete data objects such as files or messages. As Figure 3.5 shows, a correspondent host starts the process by uploading an object to a storage service, the mobile host queries this same service at some point in the future, and downloads the object if it wishes to. There is no direct contact between the two hosts, so the mobile host must know where the object is being stored, and check for updates periodically. This approach suits mobile hosts well even when a system is not designed specifically for them, because the recipients initiates the final download, and thus there is no need for the system to maintain knowledge of current host addresses.

Many widespread Internet message or file delivery mechanisms are storage-based. In electronic mail (or email), a correspondent user uploads a message to a known mail server, with a destination of the form *user@domain*. This server delivers the message to the destination server, whose IP address is obtained by resolving the domain portion of the address through a DNS lookup. The destination server stores the message until the specified user authenticates themselves and retrieves it. Web and File Transfer Protocol (FTP) servers are two other common content distribution mechanisms. Files uploaded to the server's file system become available to other users across the Internet, potentially restricted to only those who are authorised. Content is not addressed to a specific user, but they may either discover the content by browsing the server, or be alerted to its presence via a URI obtained by some other means.

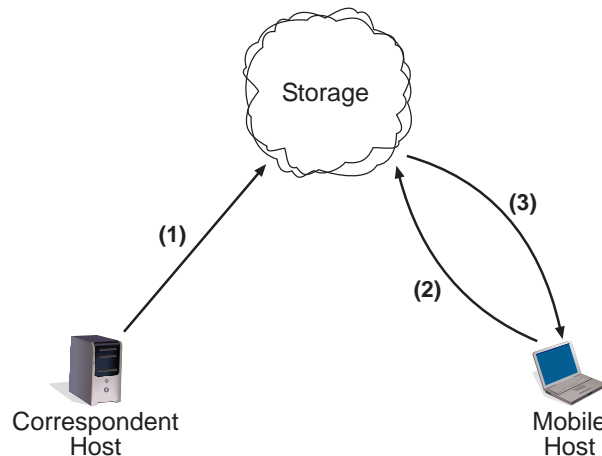


Figure 3.5: Storage-based communications: (1) the correspondent host sends the data item to a storage location; (2) the mobile host checks for new data, and (3) retrieves any that is available.

The storage concept also extends to distributed file systems. These present hosts with persistent storage that acts as an extension to their local file system, but is actually on a remote system. Files placed in this shared namespace are then readily available to other hosts. In some instances, such as the Networked File System (NFS) (Shepler *et al.*, 2003) and Coda (Kistler & Satyanarayanan, 1992), the file system is hosted upon one or more dedicated servers, which the hosts are configured to connect to. Coda supports mobile hosts by way of disconnected operation, where selected files are cached locally so that they can be accessed if network connectivity is lost, and any changes made to the cached copy are integrated in to the network copy on reconnection. There are also a number of peer-to-peer distributed file systems, where the hosts that use the system cooperate to store the files in ways that maximise availability. These include Farsite (Adya *et al.*, 2002), which is coordinated by way of a hierarchical directory service, and OceanStore (Kubiatowicz *et al.*, 2000), Ivy (Muthitacharoen *et al.*, 2002) and the Cooperative File System (CFS) (Dabek *et al.*, 2001) which are all built upon structured peer-to-peer overlays. A DHT alone can also be used to store objects within a flat namespace, at a location derived from some property, for later retrieval by a host that also knows or is interested in the same property.

Tuple spaces, as exemplified by Linda (Gelernter, 1985), enable hosts to communicate through tuples placed in a persistent shared repository. Hosts insert data tuples in to the repository, or attempt to read tuples that match a given pattern (choosing whether or not to remove the tuple from the repository in the process). If there is no matching tuple when the read request is placed, then it remains active until one does become available, at which point the requesting host receives the data immediately.

Generally, the closest these systems have to addressing is shared knowledge of where an object is stored. If one host wants to reach another, then it uploads the data object to a particular server, file path and/or identity that it knows the other host will look for. There is also the potential to reach numerous other hosts beyond those that were the original target.

These approaches are suitable when delivering large amounts of data, particularly when the destination device experiences large periods of disconnection. This is because the storage location will usually be willing to hold the content for a long or even unlimited amount of time, such that it remains available until the device can collect it.

3.3.4 Opportunistic networking

Opportunistic networking is an emerging research area that presents an extreme form of asynchronous messaging. It operates in an environment where there is no guarantee of a complete path between the sender and receiver, and where message delivery may experience long delays. There are various systems built on these principles, but the common mechanism is that hosts harness device mobility to deliver messages between networks where no other communication medium exists. Devices may be moving because people are carrying them (Davis *et al.*, 2001), because they are attached to vehicles (Burgess *et al.*, 2006), or even because they are attached to animals (Juang *et al.*, 2002). Some systems use opportunistic networking to deliver messages in a purely ad hoc environment, but this section concentrates on those that enable Internet-scale communication. This covers such situations as in Figure 3.6, where one mobile device transfers a message to another device, which later forwards the message to the destination across the Internet. The survey by Pelusi *et al.* (2006) covers a number of different opportunistic networking systems in detail.

Haggle (Scott *et al.*, 2006) is a framework that enables devices to transfer data in three different ways: over the Internet, between devices, or by physically moving to another location. Which option they use depends upon who the data is intended for, and what options are available at the time. A device that wishes to send a message encapsulates it with associated metadata in an Application Data Unit (ADU), which is a sequence of key/value pairs. The device adds a number of destination addresses to the ADU, specified in any protocol that it has a supporting module for. These addresses are typically user-level identities that make sense for Internet-scale transfers, like email addresses or phone numbers, but can also be device specific identifiers like MAC addresses.

Once the message has a destination, the device stores it ready for forwarding. If the device's movements bring it in to contact with the destination, it can deliver the data directly.

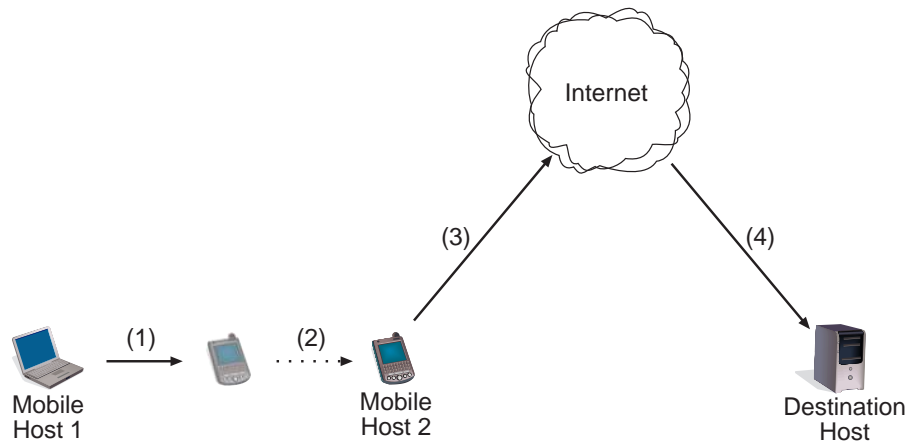


Figure 3.6: An example of opportunistic networking: (1) Mobile Host 1 forwards a message to Mobile Host 2; (2) Mobile Host 2 moves; (3) Mobile Host 2 gains Internet access and forwards the message; (4) the destination host receives the message.

It can send the data over the Internet if it connects to an access point and the ADU contains an Internet addressable identity. Otherwise, the device can forward the message on to other Huggle devices that it encounters. The intention is that the movements of the other device may eventually bring it in to contact with the destination, or an Internet access point, or another device that can take the ADU even closer to the destination. While Huggle does not have any specific method for contacting other mobile devices across the Internet, it could be used in conjunction with any of the techniques covered previously in this chapter.

DakNet (Pentland *et al.*, 2004) uses mobile access points mounted on vehicles to ferry data items such as emails or audio messages between kiosks in remote rural communities and Internet hubs in larger towns. It has been deployed in areas of India and Cambodia, with buses, motorcycles and even ox-carts carrying the access points. It provides villagers with the ability to send and receive Internet content in areas where it would otherwise be very difficult to obtain access.

These systems share similarities with Walkabout, as devices with limited connectivity hold data messages until they have the opportunity to forward them closer to their destination. These opportunistic networking solutions make use of inter-device forwarding in addition to Internet transfers. However, they do not have the specialised Internet mobility support that Walkabout proxies provide to maximise a device's potentially limited Internet access. Huggle supports many different Internet transfer methods, and could conceivably interact with the Walkabout architecture via an appropriate protocol module.

3.4 Proxies

As defined in Chapter 1, a proxy mediates communication between a device and the wider network. A proxy's positioning allows it to perform data storage, redirection and transcoding tasks on behalf of a device. These tasks are of particular importance if the device is mobile, because they help to improve the amount of data that the device can send or receive during its limited connection periods. This section explores some ways that proxies are used to support mobile devices.

A caching proxy like Squid (Squid, 2007) typically resides within a local network, and intercepts all requests for web content made by client devices. It services these requests by retrieving data from the Internet as necessary. However, if the requested content is already in-cache, the proxy delivers it to the client immediately. This local delivery yields faster download times for the clients, as well as reduced Internet usage for the individual or organisation that is running the proxy.

Mowser (Joshi *et al.*, 1997) is a local caching proxy that adapts content to suit the capabilities of mobile devices. The user configures Mowser via a web form, then browses the web as normal. The proxy then transforms any data that it delivers to the client to meet these criteria. Reducing the fidelity of images, video and audio decreases the content file size, but may not be noticeable on a simple mobile device. Therefore, these transformations are an effective way to increase the amount of data that a device can retrieve during a limited connection time, particularly when using a slower transfer method such as Bluetooth. Buchholz & Schill (2002) present an architecture based upon the same principle, where a hierarchy of proxies exist within the Internet. A mobile device includes a description of its media handling capabilities within any web request it makes, and the proxies communicate to deliver an appropriately adapted data object to the device. The architecture by Kara & Edwards (2003) uses local proxies to support streaming multimedia retrieval on mobile devices. Proxies download and forward segments of a stream while the device is connected, and other proxies selected by the Mobile Motion Prediction algorithm (Liu & Maguire, 1995) pre-emptively download pieces, with the aim of providing an uninterrupted service to the device.

Proxies are also useful when data is sent to a mobile device, rather than requested by it. As shown by several of the redirection-based systems from Section 3.3.2, a proxy acting on behalf of a mobile device can allow it to participate in messaging systems that were originally designed for immediate delivery to fixed hosts. For example, Warp (Zhao *et al.*, 2004) gives mobile devices access to a Tapestry DOLR service, while the work of Caporuscio *et al.* (2003) allows devices to receive pub/sub notifications that are sent in their absence.

3.4.1 Infostations

The Infostations architecture (Frenkiel & Imielinski, 1996; Goodman *et al.*, 1997) uses access points with caching functionality, known as *infostations*, to provide high speed data transfers to mobile devices in a disconnected environment. The motivating idea behind the research is that data bound for a mobile device is usually delay tolerant, so high-speed transfer bursts in between periods of disconnection can be at least as effective as a service that provides low speed ubiquitous coverage. Infostations achieve this with low power demands on mobile devices and at a lower cost than cellular data transfers.

Each infostation has a customised MAC layer that maximises throughput by only communicating with a single nearby device at a time. Mobile devices use them to retrieve data from the Internet or a private network. Infostations might be provided as a service by a single company, which manages the transfers between them over dedicated lines. They may also be more ad hoc, with organisations or individuals providing their own infostations that connect to each other across the Internet. Or they could be used solely for local communications, within an office, campus or the home. The quality of the service depends on suitable placement of infostations, particularly at regular intervals, and the authors suggest deployment in such places as toll booths, street corners, building entrances, train stations and airport waiting areas.

The majority of Infostations research concerns situations where downloads from the network to a user's device form the majority of the data flow. The basic operation is for a device to request data it wants when it has access to an infostation, and for the infostation to forward data packets as they arrive. It may take multiple connections for the device to receive the entire requested item. The latency between the request and the data arriving can affect efficiency, particularly when the device is highly mobile. One solution is to have separate upstream and downstream infostations available, to enable devices to place their requests even if another is currently downloading (Goodman *et al.*, 1997). Another approach is to use a low-bandwidth cellular network to place requests, so that the data is locally available for download when the device reaches an infostation (Frenkiel *et al.*, 2000).

A cluster controller (Iacono & Rose, 2000) can coordinate data delivery to the individual infostations, by acting as an additional proxy between a group of infostations and the rest of the network. These controllers attempt to minimise the overall delay by delivering packets to an infostation in advance, for immediate delivery when the device enters the coverage area. If a device's path is very predictable, such as when travelling along a highway, the controller can forward different packets to the infostations along its path well in advance. However, the work also presents an algorithm to handle more complex movement scenarios. The controller

considers how much of the data item remains and which infostations the device could reach following a constant velocity random walk, then uses this knowledge to spread packets across the infostations in the device's likely path, reducing delay at the expense of additional network traffic.

One adaptation of the Infostations architecture provides on-demand access to multimedia content broadcast over DVB (Bush *et al.*, 2005). Each infostation builds up a catalogue of broadcast material and transcodes it to suit a range of possible devices. A user retrieves a listing of the available files when they enter the coverage area of an infostation, select what they wish to download and pay for it if necessary. They continue to download parts of the file whenever they are connected to an infostation, until they have it all. The infostations are responsible for accepting payments and authorising data requests.

Infostations and Walkabout share similar goals of maximising throughput while a mobile device is connected, although the Infostations architecture is concerned with the retrieval of requested data, rather than the delivery of unanticipated messages. Infostations use an efficient MAC protocol and pre-emptive packet delivery to improve transfer rates. Walkabout supports large data uploads in addition to downloads, and it could potentially benefit from using the same MAC layer in its access points as infostations do.

3.5 Mobile device movement prediction

Movement prediction can be used to improve various aspects of systems that deliver data to mobile devices. It can help to reserve network capacity in advance, aid real-time call admission control, or deliver data to a location before a mobile device connects. The last ability is particularly important for this thesis, because an existing cache of data can lead a mobile device to experience faster download speeds upon reconnection. For example, prediction has been used previously to deliver packets to infostations along a device's likely path (Iacono & Rose, 2000), to migrate pub/sub subscriptions and notifications (Burcea *et al.*, 2004), and to pre-emptively cache media streams (Kara & Edwards, 2003).

A common assumption is that a person's past movements give insight in to their future movements, because they tend to follow a number of regular paths on a daily basis. Therefore, many prediction algorithms compare a device's recent movement sequence to a record of its historical sequences, where each item in the sequence is a discrete action, such as a connection to a cellular base station or a WiFi access point. Previous approaches that rely upon historical evidence have used Lempel-Ziv algorithms (Bhattacharya & Das, 2002),

Markov predictors (Song *et al.*, 2006), Bayesian Learning (Akoush & Sameh, 2007) and the Mobile Motion Prediction (MMP) algorithm (Liu & Maguire, 1995). The survey by Cheng *et al.* (2003) covers the details of a number of different methods that are used to predict device movement in wireless systems.

The extended Walkabout model uses Markov predictors to improve transfer performance in Chapter 6. An order- k (or $O(k)$) Markov predictor compares the last k items of a device's recent movement context against its entire recorded movement history, to predict where it is most likely to move to next. If the recent pattern exists in the history, the location that follows on the most occasions becomes the prediction for next step. If the pattern has never occurred previously, then no prediction can be made. The use of lower-order *fallback* can improve the chance of obtaining a result, where $O(k-1)$ predictors are applied recursively until a location can be predicted, or the context is exhausted. When applied to a large corpus of WiFi access point traces, $O(2)$ Markov prediction with fallback was found to have a median prediction accuracy of 72% (Song *et al.*, 2006).

3.6 Summary

This chapter has introduced all of the background material that is necessary to understand the upcoming chapters of this thesis, by covering the areas of service discovery, the use of proxies, peer-to-peer overlay networks, and how to locate, maintain communication with, and predict the movement patterns of mobile devices. In doing so, it forms Contribution C1 from Section 1.4.

While the complete design of Walkabout is different to all of the systems presented here, it does share partial similarities with some of them. In particular, Walkabout's data downloads are similar to those enabled by Infostations, uploads are similar to Huggle, and Internet transfers are influenced by BitTorrent.

The following chapters discuss the full design of Walkabout, and build upon the research and insights that were presented in this background review. Experimental results show that an appropriate combination of the principles presented here can create an architecture that makes it practical for mobile devices to participate in Internet data transfers.

3.6. *Summary*

Chapter 4

Core system design

The main focus of this thesis is Walkabout, a messaging architecture designed to support asynchronous data transfers where at least one of the participants is likely to be a mobile device. It enables a device to send a message across the Internet to one or more other devices in a robust and efficient way that maximises the use of network connection time available to them. This chapter begins by giving a brief overview of how Walkabout works and its key features. It continues by describing the components of Walkabout and the processes that they follow to carry out these transfers, then it covers the issues of reliability, security and real-world deployment. Finally, it discusses an implementation of the architecture and a sample application, and presents the API that they use.

4.1 Overview

The Walkabout architecture aims to make it practical for mobile *client* devices to send and receive messages of any size across the Internet. A client communicates directly with a Walkabout *proxy* that is ideally located within the same network segment. The proxy participates in peer-to-peer overlays on top of the Internet, to communicate with other proxies and carry out message transfers on behalf of its clients.

A complete transfer follows several steps. First, the *producer* client generates a message, then uploads it whenever it is connected to a proxy. The proxy creates a *message tracker*, then uses location-independent addresses to contact the last known proxy for each *consumer* client. These proxies become *peers* to each other, and the links between them form the message delivery overlay. Peers retrieve pieces of the message over these links, then forward them to the consumers. Figure 4.1 illustrates what an overlay constructed to deliver a message to three

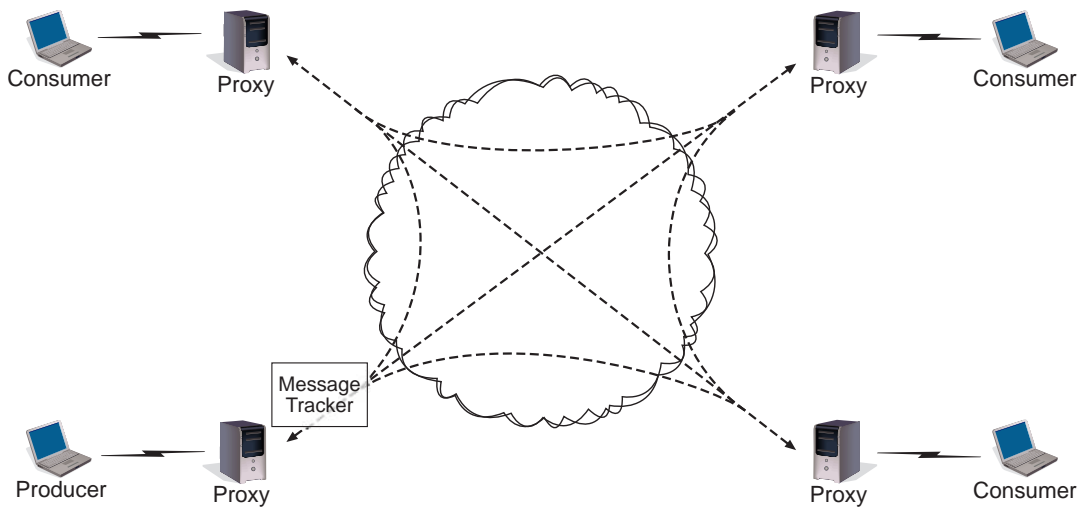


Figure 4.1: A Walkabout overlay network.

consumers might look like. Each overlay is dedicated to the delivery of a single message, and a proxy can participate in any number of overlays at once. As Section 3.2.2 explains, this style of peer-to-peer delivery offers reliability and speed improvements compared to transferring directly or via central storage, and allows anybody to distribute large amounts of data without access to a high performance server.

When all of the consumers have received or rejected the entire message, the transfer is complete. This store-and-forward approach makes delivery robust when clients are mobile. This is because proxies accept pieces of the message from mobile producers, and continue to forward them even if the producer disconnects. Proxies also store pieces of the message when a mobile consumer is unavailable, then forward them immediately when the consumer reconnects.

Walkabout nodes, be they proxies or clients, communicate by way of sending small *notifications* to each other. Note that this is distinct from a message, which is typically much larger and transferred between nodes via a series of notifications.

This design combines the redirection and storage-based Internet communication methods from Chapter 3. Clients send messages to each other using location independent addresses and proxy overlays redirect them to the correct destinations, even if the consumers are moving. However, the messages can be large, so the overlays do not just send them across the Internet automatically. Instead, the proxies store pieces of the message and their peers retrieve them on demand for delivery to consumers. This hybrid approach gives devices a flexible way to send and receive messages while moving without placing undue stress on the intermediate network.

The roles of the system components are explained in more detail in Section 4.2, and the full details of the transfer process are explained in Section 4.3.

4.1.1 Features

The core design features of Walkabout that this chapter covers are as follows:

- *Location independent addressing:* A producer addresses a message to a consumer using a location independent key. The key resolves to the consumer's last known location via a standard Distributed Object Location and Routing (DOLR) service.
- *Multiple recipients:* A message can be delivered to a group of consumers in a scalable manner.
- *Easy access:* If a device can find a Walkabout service in the local network, it is ready to transfer, without needing to worry about firewalls or network address translation. The software to provide the service locally can be installed on any fixed host.
- *Decentralised delivery:* Overlay creation is ad hoc and managed by the initial peer. If that peer fails, then another one takes over. Message transfers take place directly between peers.
- *Message segmentation:* A message is broken in to pieces during transit, so that a transfer may be interrupted and resumed as necessary. Segmentation also allows multiple proxies to provide parts of the same message simultaneously, so that they can be downloaded in parallel to accelerate transfers.
- *Unrestricted message sizes:* Equal support for all message sizes, ranging from very small (several bytes) to very large (gigabytes).
- *Upload caching:* A producer may upload a message to the Walkabout overlay as quickly as its local network allows. If the uplink speed across the Internet is slower, then the local proxy caches the message pieces and transfers them across the overlay at a later time, as fast as the connection allows.
- *Download caching:* The overlay transfers each message piece across the Internet to the proxy where the consumer connected last. If the consumer is still present, the proxy delivers the piece immediately. If the consumer is not present, the proxy caches the piece until it reconnects to the network.

There are also additional design features that Chapter 6 covers in depth:

- *Pre-emptive message delivery*: By predicting movement patterns, messages are distributed in advance to improve download times.
- *High level addressing*: Multiple devices and multiple users can be united under a single address, and the address owner specifies rules to direct messages to the end devices.
- *Simple device support*: Non-programmable devices that are unable to run Walkabout software can still send and receive content.

4.2 Components

The main components of Walkabout are clients, proxies and message trackers, which are supported by DOLR and service discovery. This section explains the main properties of these components.

4.2.1 Distributed object location and routing service

Walkabout could potentially be deployed in millions of networks around the world, so it requires a scalable mechanism for proxies to contact each other on demand when seeking clients and message trackers. This is something that a Distributed Object Location and Routing (DOLR) service such as Tapestry (Zhao *et al.*, 2001) can provide. As Section 3.2.1 explains, DOLR enables a host to publish an identifier, which other hosts can then direct notifications to. Therefore every proxy joins a common DOLR service when it launches.

A proxy publishes a new unique identifier when it either accepts a client registration or creates a new message tracker, which is based upon the hash of the client's public key or the hash of the message metadata, respectively. Any other proxy that wishes to make initial contact can then do so by "routing" a notification to that identifier. Hereafter, a notification is said to be *routed* when it is sent via the DOLR service. When a client moves or the message delivery is complete, the proxy unpublishes the appropriate identifier.

4.2.2 Service discovery

Upon connection to a new network, a client needs some way to automatically find a local proxy. Alternatively, if there are no proxies available locally, then they should be able to find one in a remote network instead. While there are a number of potential service discovery protocols presented in Section 3.1, Apple's Bonjour integrates local and remote discovery

functionality in to the one technology (Apple Computer, Inc., 2006). Local discovery under Bonjour is decentralised, so a client should always be able to use it to find a local proxy if it exists, without the need for any support infrastructure, and regardless of what other service discovery protocols are in use in the same network.

Walkabout therefore requires each proxy to run a multicast DNS responder (Cheshire & Krochmal, 2006b), which is the client-side component of Bonjour. Proxies are always available locally, but administrators also have the option to register their proxy with a DNS server that supports dynamic updates, if they are willing to open the service to external connections. The DNS server is not required for the system to function, but it does make it more flexible by allowing a client to use Walkabout even if there is no local proxy available.

4.2.3 Client

A client is any device that runs an application to send or receive Walkabout messages. A producer client initiates message transfers, and addresses them to be received by one or more consumer clients. While Walkabout is designed with mobile clients in mind, it is also equally suited to fixed systems.

Each client has a key pair that it uses as the basis for identification and cryptography. In particular, the hash of the client's public key (known as the *key hash*) serves as a globally unique identifier, which a proxy uses to register the client with the DOLR service upon connection. This allows other proxies to route notifications to the client's last point of attachment by simply providing the key hash.

4.2.4 Proxy

A Walkabout proxy is a network service that provides client connection, message transfer, caching and overlay maintenance functions. This service would typically be offered by a program running on a single dedicated machine within a local network. It could be provided as a free service to trusted users in a home or office environment, as additional value to accompany other services (in a café, for example) or as a wide scale subscription service deployed alongside wireless access points. As mentioned previously, proxies use Bonjour to advertise their service on the local network and, optionally, for remote access across the Internet.

Proxies link to form peer-to-peer overlay networks for the delivery of messages. They receive message pieces for upload from clients, and download them from their peers. Any

pieces they acquire are held in a cache as long as possible, so that they may be delivered to clients connected locally, or downloaded by any other proxies that may need them. When the cache reaches its limit, the proxy deletes pieces according to the cache purging policy, which can be selected to suit the proxy's capabilities.

A single machine can run both a client and a proxy, though this is only suitable for fixed systems that are unlikely to disconnect, such as desktop computers.

4.2.5 Message tracker and the overlay

The collection of proxies that have or seek pieces of a given message form the peers of a Walkabout message delivery overlay. Each overlay is managed by a message tracker, which stores peer addresses and monitors the delivery status of message consumers. When a producer starts uploading a message for the first time, the proxy it is connected to creates a message tracker as a local process, and publishes its location to the DOLR service using the hash of the message metadata. Other proxies register with the message tracker, and thus join the overlay, as they come in to contact with the producer or one of the consumers. They also update the tracker when a connected consumer either finishes downloading the message, or chooses to reject it. The overlay only exists to deliver the message, so once all the consumers have received or rejected the message, the tracker notifies the producer and all the peers, and the overlay is dismantled.

4.3 Message delivery process

Any time that client acquires a network connection, it uses a service discovery protocol to seek out and register with its nearest proxy. When a client wishes to send a message, it uploads a header to its proxy that describes the message and the intended consumers, then continues by uploading pieces of the message as fast as the local network will allow. This proxy creates the message tracker that will manage the delivery overlay, then contacts the last known proxy for each consumer. In response, each proxy registers with the message tracker then contacts all of the other proxies that are already registered. Proxies communicate over this peer-to-peer overlay, and download the message from each other on behalf of the consumers. A proxy caches any message pieces that it downloads from its peers, and forwards them on to any consumer that is connected. Proxies continue downloading message pieces and delivering them to consumers until all of them either have the message or haven chosen to reject the delivery.

This section describes how a client connects to a proxy, covers the message delivery process of message generation, producer upload, overlay transfer and consumer download, then concludes by explaining what happens when delivery is complete.

4.3.1 Client connection

A client needs to find a proxy to gain access to the system, so upon joining a new network, it initially performs a local Bonjour search. If it is able to find a suitable proxy, it initiates a connection. If there does not appear to be a local proxy, it instead uses Bonjour to search a pre-configured domain over the Internet, in an attempt to find one in a remote network. While it is preferable to use a local proxy, it is unlikely that one will be available in every network, so a remote proxy may be the only option at times. This does introduce problems of needing to negotiate firewalls and address translation, so it may not always be possible for the client to search for or make contact with an external proxy.

Upon connection to the new proxy, the client issues a notification that contains its key hash, the address of the last proxy it was connected to, and information about any outstanding messages that it is a producer or consumer for. The proxy uses this information to carry out a number of tasks:

- It publishes the client's key hash to the DOLR server.
- It contacts the client's previous proxy, to let it know the client has reconnected. That proxy revokes its existing DOLR publication for the client, and responds with the headers for any new messages that were addressed to the client in its absence.
- If it has any locally cached pieces of the messages the client has expressed interest in, it delivers them immediately.
- It joins or creates the appropriate overlays to upload or download messages on behalf of the client, as informed by headers it received from the client or previous proxy.
- It routes a delivery status query to the message tracker for any message the client indicates it has fully uploaded, and returns the result to the client.

There is a slight chance that a new message could be routed to the wrong proxy in the brief period between when a client reconnects and its previous proxy unpublishes its identity from the DOLR service. The previous proxy knows the correct proxy by this point, so it forwards the details on directly to the correct proxy if this happens.

Table 4.1: Message header fields.

Field	Description
Cryptographic information	Identifies the hashing and symmetrical encryption algorithms used (if any)
Message signature	The hash of all the other fields combined
Payload length	The total length of the data payload (in bytes)
Piece length	The fixed message piece length (in bytes)
Piece signatures	An array of individual piece hashes
Producer key hash	The identity of the producer
Consumer key hashes	The identities of the intended consumers
File name	The default name if the message is written to a file (optional)
Application data	Any additional information the application wishes to communicate to the consumers (optional)
Session keys	Encrypted symmetrical cryptographic keys used to secure delivery (optional) [see Section 4.7]

4.3.2 Message generation

The delivery process begins when a client application generates a new message, which consists of a data object as a payload, and associated metadata stored in a *header*. The payload is either drawn from a file on disk, or is the direct output of a program. The application divides the payload into equal length pieces at the time of selection, where the length is either chosen automatically or specified by the user, and remains fixed for the duration of the transfer. The header describes the payload and its destination(s), as Table 4.1 shows. Most of the fields are straightforward, with the exception of the piece and message signatures. The piece signature array contains the hash of each individual piece of the payload, which provides proxies and clients with a way to check the validity of pieces after downloading them. The message signature is the hash of all of the other fields in the message header. Once created, a header is immutable, so this hash uniquely identifies the message. It is therefore used throughout the architecture to refer to the message, and as the DOLR address that identifies the message tracker. Because the entire message needs to be known to calculate the header, all transfers must be discrete messages, and therefore Walkabout does not natively support streaming data. See Chapter 8 for a discussion of how changes to Walkabout could enable it to support streaming data. The header is an important part of the delivery process, because it contains all the information that a proxy requires to create a new overlay, or join an existing one.

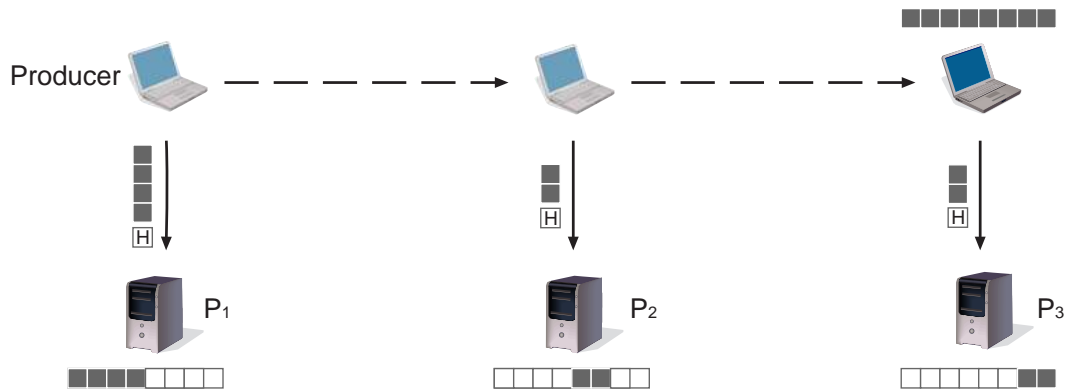


Figure 4.2: A producer continues uploading a message as it moves between proxies.

The various decisions regarding when to generate the message, what it should contain and who to send it to may be either user-driven or automated, depending upon the type of application and the capabilities of the device. For example, an application running on a camera could generate a new message each time a photo is taken, and address it to a pre-configured backup server. Alternatively, a user interacting with an application running on a laptop might use dialogue boxes to select files from the local file system and consumer aliases from a list, where that list has been compiled through out-of-band means, such as over email or from a directory service.

The basic data structure used to communicate message state is a *piece map*, which is a string containing as many bits as there are pieces in the message. The meaning of a bit depends upon the context. If a client delivers a map, a bit is set if they want the piece in that position, while a proxy uses this to mean that they have that piece in their cache. These maps are generally quite small. For example, a one gigabyte file transmitted as 256KB pieces generates a map of only 512 bytes.

4.3.3 Producer upload

After a producer has established a connection to a proxy, it is ready to begin uploading its message. It begins by uploading the header, then message pieces as fast as the connection will allow. Each time a piece upload is complete, the proxy verifies it against the hash in the header and acknowledges it if it is valid. This continues until the entire message is uploaded or the producer disconnects. If a transfer is interrupted through disconnection, the producer sends the header again upon reconnection and resumes the upload from the first unacknowledged piece. Figure 4.2 shows this process, where the producer has the full message, and uploads the header and a number of pieces to each of the proxies it connects to.

If the proxy has not seen the header before, it attempts to join the overlay by using the header's message signature to route a registration notification to the message tracker. If it exists, the tracker responds with a list of peers, and the proxy contacts each of them in turn. However, if no tracker is found, the proxy creates a new one, adds itself as the initial peer, and publishes the message signature. It routes the message header to the key hash for each of the consumers and the transfer process is ready to begin.

Once the producer believes it has completely uploaded a message, it checks with its proxy. If the proxy knows that at least one copy of each piece exists across the overlay, it informs the producer that the upload is complete. If there are any pieces that cannot be found (which may happen if the peer that held them disconnected unexpectedly), the proxy informs the producer that it must upload them again. Similarly, if pieces are lost at any point after this, the producer is also prompted to upload them once more.

Producer applications usually need to know when a message has been successfully delivered. For example, a backup application needs to know when a file is safely in the archive so that the local copy can be deleted. Walkabout achieves this by having the proxy at the receiving end update the message tracker when it completes a transfer to the consumer. The tracker then sends a status update to the producer, which it receives if connected. If the producer misses the update, it checks on the delivery status each time it reconnects until all consumers are accounted for.

4.3.4 Overlay transfer

Peer-to-peer file transfer protocols such as BitTorrent (Cohen, 2003) form the inspiration behind the message transfers across a Walkabout overlay. Transfers are driven by receiving proxies and make use of parallel downloads where possible. As a proxy downloads the message from its peers (or receives it from the producer), it keeps all peers informed of its progress, so that they maintain knowledge of message availability and are able to make informed download decisions.

While data transfers between clients and proxies are performed in pieces, transfers between proxies are performed in blocks, which are a fraction of the size of a piece. Figure 4.3 illustrates the difference. This design uses 16 blocks per piece, which is the same value that BitTorrent has shown to be suitable for peer-to-peer transfers. These smaller units increase the number of objects available in the overlay, and thus increase the chance that a proxy can download from multiple peers in parallel. Progress updates between peers are still at the granularity of pieces, though, so that the size and frequency of updates are kept to a minimum.

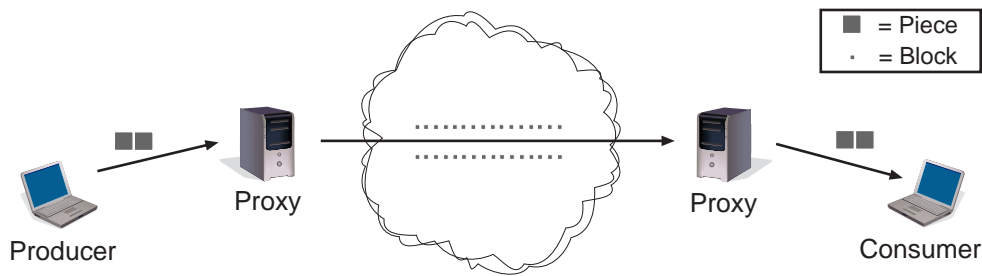


Figure 4.3: The use of pieces versus blocks.

The transfer negotiation begins when one proxy receives a message header from another. The proxy starts by attempting to find which pieces (if any) its connected consumers want. It forwards the header to each of them, and expects a piece map detailing which pieces they want in reply (which may never happen if the client has disconnected). It initially assumes that they want the entire message, and starts downloading blocks immediately from the proxy that delivered the header. If the client responses indicate that there are any pieces that none of them want, the proxy shifts its focus to avoid downloading them. A consumer may also choose to reject the transfer completely, which leads the proxy to route a rejection notification to the tracker, and possibly to cease downloading the message itself. Once again, the device capabilities and the particular application determine whether the message acceptance is decided automatically or through user input.

The next step sees the proxy contact the message tracker to join the overlay. It retrieves the list of peers, requests a piece map from each one to find what pieces they can offer, then starts downloading blocks simultaneously from multiple peers (where possible). Proxies pipeline block requests by requesting several at once from each peer, which ensures that a block is being delivered while a new one is being requested whenever possible.

Which pieces, and therefore which blocks, a proxy requests from its peers is determined by a set of rules with two main aims: to minimise total download time, and to minimise the individual piece download times. The full set of rules are outlined in Algorithm 1, which is expressed in the Python programming language (Python, 2007). The proxy sends consecutive requests until it reaches the maximum number of block requests per peer, or no more suitable blocks remain. When one download completes, a new block is requested immediately.

Minimising the transfer time between a single producer and consumer is quite straightforward: the consumer-side proxy fills the pipeline with requests, and the producer-side proxy delivers them as fast as the link will allow. When a message is addressed to multiple consumers,

4.3. Message delivery process

Algorithm 1 Piece and block selection.

```
1 # Find which piece and block of message 'sig' is best to download from 'peer'
2 def getNextBlock(sig, peer):
3     remaining = ~self.cachedMap[sig] # Invert the map of cached pieces
4     needsMap = remaining & self.clientNeeds[sig] & peer.getMap(sig)
5     needsList = needsMap.listPieces() # List all that are still needed
6
7     # Start by choosing from the unique pieces this peer can offer
8     uniqueList = (needsMap & peer.getUniquePiecesMap(sig)).listPieces()
9     if not empty(uniqueList):
10        last = peer.lastPieceRequested(sig)
11        if last in uniqueList and hasUnrequestedBlock(last):
12            return (last, nextUnrequestedBlock(last))
13        random.shuffle(uniqueList)
14        for piece in uniqueList:
15            if hasUnrequestedBlock(piece):
16                return (piece, nextUnrequestedBlock(piece))
17
18    # Choose from pieces that already have blocks requested from any peer
19    partialMap = needsMap & findPartiallyRequestedPieces(sig, self.peers)
20    partialList = partialMap.listPieces()
21    partialList.sort(leastRecentlyRequestedFirst)
22    if not empty(partialList):
23        return (partialList[0], nextUnrequestedBlock(partialList[0]))
24
25    # Choose from the pieces with the highest availability in the overlay
26    commonMap = needsMap & findMostCommonPieces(sig, self.peers)
27    commonList = commonMap.listPieces()
28    random.shuffle(commonList)
29    for piece in commonList:
30        if hasUnrequestedBlock(piece):
31            return (piece, nextUnrequestedBlock(piece))
32
33    # Finally, just pick from any piece the peer has at random
34    random.shuffle(needsList)
35    for piece in needsList:
36        if hasUnrequestedBlock(piece):
37            return (piece, nextUnrequestedBlock(piece))
38
39    return (None, None)
```

though, the producer-side proxy becomes a potential bottleneck. It is initially the only source for all of the pieces, so the download can not complete for any consumer until it has uploaded every piece at least once. If the proxy uploads any piece multiple times before this point, the system-wide transfer speed suffers. Accordingly, the consumer-side proxies identify when a peer has any unique pieces available, and prioritises them above any other pieces they have on offer, in an effort to improve the rate at which unique pieces are injected in to the overlay. This can be seen in lines 7–16 of Algorithm 1.

The other goal, of minimising the download time for each piece, is aimed at improving transfers to mobile consumers. Simply downloading a large number of blocks while a consumer is connected is not sufficient. For blocks to be delivered to the consumer, they must form complete pieces, so the proxy focuses on completing as many pieces as possible over the duration of a consumer’s potentially limited connection. When selecting the next piece from a peer, it first favours those that it already has some blocks for, beginning with those that were started furthest in the past (lines 18–23). Next, it selects from the pieces that are most widespread across the overlay, and will therefore be available to download in parallel from the maximum number of peers (lines 25–31). If all pieces remain equal after that, it selects one at random (lines 33–37), or nothing at all if every block they can offer has already been requested.

Once a proxy collects all 16 blocks of a piece, it verifies the piece against the hash value in the header. It stores the piece in its cache if verification is successful, sends an update to each of its peers, and delivers the piece to any connected clients that want it. Should the verification fail, whether due to a transmission error or a malicious peer, then the entire piece needs to be re-downloaded. The simplest way for a proxy to communicate message progress to its peers is to send out a piece map each time it completes a piece. While this is an efficient way to communicate how much of the message is in the proxy’s cache, consecutive updates can contain large amounts of redundant information when there are only minor differences between them. Thus, proxies actually use a *new piece list* for most peer updates, which contain the numbers of the pieces that have been acquired since the last update. Piece maps are still used, though, for the first time that a proxy updates a new peer, when the peer explicitly requests one, or when there are enough new pieces that the size of the new piece list would actually exceed that of the piece map.

The selected piece size, and thus block size, can have a dramatic effect on the performance of a transfer. Small pieces increase the chance of parallelism, but the volume of requests and progress updates can slow the rate of message data transfer, and the size of these overheads increase with payload size. By contrast, big pieces reduce these overheads, but the

corresponding drop in parallelism can cause speeds to suffer when a message is sent to multiple consumers. There are also issues to consider regarding the transfers between client and proxy. The effect of piece size on the overall transfer performance is explored in Chapter 5.

The complex flow of message headers, pieces and blocks when a consumer is constantly moving gives valuable insight in to the delivery process. Therefore, Figure 4.4 presents a high level view of the data flow in a scenario where a message is being delivered to a moving consumer. The sequence of events is as follows:

1. The consumer is initially connected to proxy P_2 , but then moves out of network range.
2. The producer generates a message addressed to the consumer and starts uploading all four pieces to P_1 . P_1 routes the message header to the consumer's last point of contact at P_2 . P_2 joins the overlay and requests blocks from P_1 , which is the only source.
3. The consumer connects to P_3 . P_2 stops downloading, having completed two pieces. P_3 receives the header from P_2 , forwards it to the consumer, then requests blocks from P_2 . It also joins the overlay, learns about P_1 , and requests blocks from there too. Two pieces are completed and forwarded to the consumer while it is connected.
4. The consumer disconnects again. P_3 continues requesting blocks, and completes the message.
5. The consumer connects to P_4 , and uploads the message header and its piece map. This leads P_4 to join the overlay immediately, and request blocks for the pieces the consumer still needs from P_1 , P_2 and P_3 . P_4 completes the remaining pieces and delivers them to the consumer.

4.3.5 Consumer download

The final phase of delivery is when the proxy transfers the message to a consumer. The proxy knows which pieces the consumer wants, either because it requested the information from the consumer when a new message arrived, or because the consumer provided a piece map upon connection to resume a previously interrupted download. If it already has the desired pieces of the message in its cache when a consumer connects, the proxy delivers them immediately, in order of ascending piece numbers. Once these pieces are exhausted, or if the consumer is connected when the message header first arrives, the proxy streams pieces as it receives them. The consumer verifies each piece against its hash and acknowledges its receipt

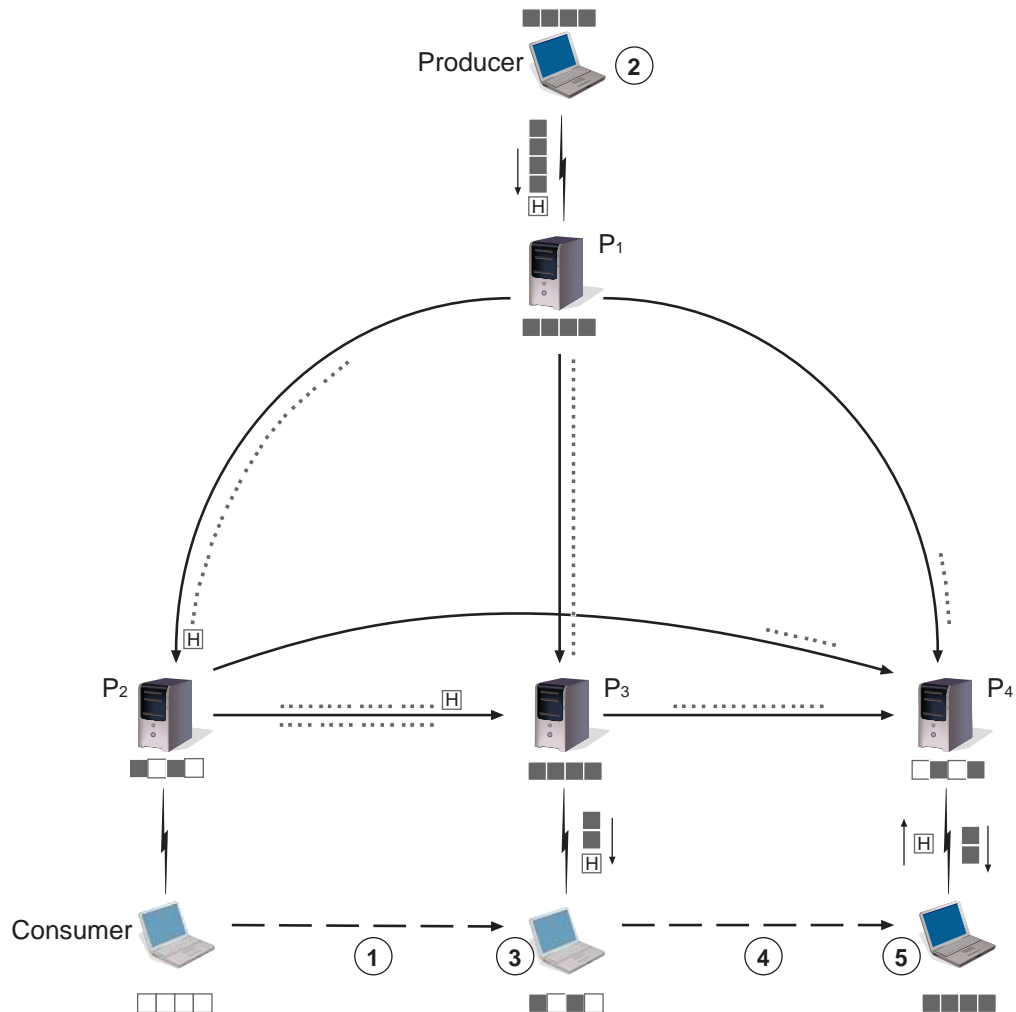


Figure 4.4: Piece and block flow in a complex delivery scenario.

if valid. The download is complete after the final acknowledgement, so the proxy routes a completion notification to the message tracker.

4.3.6 Completion

Each time a consumer completes or rejects a message, the message tracker is informed. When there are no more consumers that require the message, the transfer is complete, and the overlay can be dismantled. The message tracker sends a notification to the producer and each of the peers, and waits upon acknowledgement from them all. This causes the proxies to delete all information about the message, including any cached pieces, effectively leaving the overlay. When the producer and all peers have responded, the message tracker is deleted, and the transfer process is complete.

4.4 Cache management

The operation of Walkabout is dependent upon the use of caches, but a proxy is limited in the amount of storage space that it can offer. A busy proxy might be a member of many overlays simultaneously, causing it to store many different message pieces on behalf of many different clients. As new message pieces arrive, existing ones may need to be removed, possibly even before they have been delivered to their destination clients. This could potentially prevent a complete message from ever reaching its destination, so intelligent purging rules are required to minimise the impact of piece deletion on the system. It is worth noting that while premature deletion of pieces can have an adverse effect on overall transfer speeds, the producer can be asked to re-upload them if necessary.

Pieces that have been held in-cache for the longest and that have the highest numbers of replicas across the relevant overlay are the first candidates for deletion. The proxy notifies the peers that also have these pieces that it intends to delete them, and only does so once it receives confirmation that at least one copy will remain. If this approach does not clear enough space, then the proxy migrates pieces to its peers instead, by asking them to download some pieces with highest priority, and deleting them when the transfer completes. Any time the proxy deletes a piece, it sends out an updated piece map. There is clearly a lot of scope for optimisation, so any rules more detailed than these would require further investigation.

4.5 Short messages

Walkabout makes it simple for client applications to send messages to each other, but there are time and data overheads associated with the establishment and maintenance of a tracker and an overlay. The advantages usually outweigh these slight overheads, except when the message is small enough that the setup time is longer than it would take to just deliver the message. Therefore, Walkabout provides clients with an alternative mechanism to send short messages without the overhead of a message tracker. It requires that consumers are online to receive messages, and is primarily aimed at inter-application communication. For example, short messages could be used by a mobile device to request a file listing from a fixed server, by the server to respond, and once more by the device to request the regular Walkabout delivery of particular files.

A short message does not have an associated header like a regular message. All its information is self contained, consisting of the key hashes of the producer and the consumers, and the message payload. A message signature is derived by hashing all of these fields. The

payload can be of arbitrary size, although it is undesirable for it to be too large. The threshold that defines the maximum payload allowed within a small message is left to the discretion of the application.

To deliver one of these short messages, the client uploads it to a proxy, which routes it to the location of the consumers accordingly. The receiving proxies forward the message if the consumers are currently available, or discards it and returns a “client not found” notification if they are not. If the producer is no longer available to receive the response, it too is discarded.

Short messaging is less reliable than regular Walkabout messaging, but both mechanisms still share many features, including location independent addressing, multicast delivery and firewall negotiation. Therefore, it presents a versatile communication alternative for client applications wishing to deliver small amounts of data.

4.6 Fault tolerance

It is an unfortunate fact that network communications are imperfect. For example, transfers may not complete correctly, proxies may leave the network, or it may not be possible to locate a client. Walkabout incorporates a number of recovery techniques, to help protect against some of these problems.

When a proxy attempts to route the header for a new message to its consumers, there may be some clients that do not have an existing DOLR registration. This could happen if the consumer has never connected, if the specified key hash is incorrect, or if the client’s most recent proxy is no longer accessible. The proxy combats this by publishing the key hash itself, to become a temporary storage point for this header and any others routed to the consumer in the near future. When the consumer does eventually connect, this proxy is contacted as if it were the client’s previous proxy, which leads it to forward the message headers so that the transfers can take place as normal.

Every message tracker has a version number, which increments each time there is an update. Proxies receive the most recent version of the tracker contents when they request peer information, or in response to any update they make. This means that the tracker state is replicated across the overlay, preventing the message tracker from becoming a central point of failure. If the proxy running the original tracker fails, then the first proxy to detect this assumes its role by creating a new tracker (based upon the most recent contents it received) and publishing its existence. If it makes any future contact with a proxy that has a more recent version of the original tracker’s content, the two proxies exchange information to correct the state and bring it as close as possible to what it was previously.

A proxy maintains piece maps that describe the message availability at each of its peers, and performs periodic liveness tests to ensure that they all remain reachable. If one of these tests fails, the peer is considered to be inactive. This leads the proxy to delete its local copy of the peer's piece map and cease trying to communicate with it. It also notifies its other peers and the message tracker, which react by performing their own liveness checks. These actions mean that each peer should have full knowledge of the combined message availability at all times, so it will not take long to discover if the failure of a proxy resulted in the loss of unique pieces from the overlay. The next proxy to make contact with the producer can therefore recover these pieces by prompting it to upload them again. In addition, a proxy can use policies similar to the caching ones discussed in Section 4.4 to maintain global availability if it knows in advance that it will be disconnecting from the Internet.

Finally, as has been discussed previously, a message header contains verification hashes for each piece of its message. This makes it easy for both proxies and clients to determine when transfers have been corrupted, and to request retransmission of the damaged piece.

4.7 Security

As with any Internet-based system, security is a concern for Walkabout. Fortunately, the key pair that each client already uses for identification purposes can also enable a number of encryption and authentication techniques, which serve to secure many aspects of the message delivery process.

Because a client's key hash is drawn from its public key, the producer already has the information required to encrypt a message for each of the consumers at the time of generation. The producer generates a symmetric session key, and uses a cipher such as AES (National Institute of Standards and Technology, 2001) to encrypt the payload. It follows this by encrypting the session key with each of the consumer's public keys, and storing the sequence of results and the encryption algorithm used in the message header (as shown in Table 4.1). The piece signatures that are included in the header are calculated from the encrypted payload, which allows any host to check the validity of downloaded pieces without needing to decrypt them. Every consumer receives the same encrypted message, decrypts its copy of the session key using its private key, and then uses that to decrypt the message payload. This same approach can be used to encrypt short messages, with the encrypted session key sent as part of the message itself.

It is important that the Walkabout system is able to verify the identity of the consumers. Even though impostors are unable to access the contents of encrypted messages, there is still the possibility that they could claim a message to prevent the intended consumer from ever receiving it. Therefore, the proxy can specify in its Bonjour service announcement that it requires secure registration. This demands that the client provides its public key at the time of connection, which the proxy verifies against the accompanying key hash, and then uses it to issue a challenge-response to the client. If the client passes this test, the proxy trusts it is who it claims to be, and is willing to register it with the DOLR and forward any messages on its behalf. The client also signs all notifications it sends and the proxy checks the signatures upon receipt. These checks prevent against both denial-of-service and man-in-the-middle attacks, by preventing malicious hosts from forging or tampering with client notifications to falsely claim or reject messages.

The ability for anybody to run a proxy allows the network to grow easily as needed. Unfortunately, this flexibility also presents security problems. The ad hoc network formation means that there is no easy way for proxies to establish trust with clients or other proxies. Piece signature checks can protect against malicious proxies providing altered pieces, but not against them providing false information about their piece availability. Proxies can forward the signed notification that the client provides when it completes or rejects a message, and this allows the proxy to prove the action to the message tracker. However, there is nothing to guarantee that the proxy providing the tracker itself can be trusted! There is also the chance that rogue proxies could falsely publish DOLR entries, in an effort to prevent notifications from reaching their real destination. This can be overcome if proxies route notifications to multiple destinations (as the DOLR API in Table 3.1 details), rather than just the closest node. A potential solution to all these problems is for peers to build trust relationships with each other, as demonstrated by the TrustMe protocol (Singh & Liu, 2003) or the work of Aberer & Despotovic (2001). This is an area for further investigation, and the current design simply assumes that all proxies are honest.

There is no provision within the architecture for the distribution and management of client keys, but Walkabout would ideally be supplemented by an external public key infrastructure (PKI). This is not strictly necessary, though, as adequate functionality can still be obtained by manually copying key pairs on to devices, and distributing public keys to people through out-of-band channels like email.

4.8 Deployment

One of the advantages of Walkabout is that it is easy to deploy and access. Anybody can install a proxy, wireless Internet access is widespread, and client devices use commonly available hardware to access the service. This section outlines the important aspects of real-world Walkabout deployment.

A proxy is a piece of software that can be installed on any fixed device that is accessible through a wireless network. This could be a dedicated server machine, a desktop computer or possibly even the wireless access point itself. Organisations can deploy one or more proxies, depending on demand, alongside standard network services such as email or web access. Companies can offer them to attract potential customers, in much the same way that cafés commonly offer web access today. End-users can easily set up their own local proxy on whatever fixed hardware is available in their home. Support for remote proxy connection means that there does not even need to be a proxy in every network for Walkabout to function, although the fast transfers that local proxies provide make them preferable.

Overlay formation is ad hoc, and there is no central administration to control which proxies can use Walkabout. To become part of the peer-to-peer architecture, all that a proxy needs is access to a common DOLR service. This allows proxies to find and introduce new peers to an overlay when arranging a message transfer on behalf of a producer, or to join an overlay on demand by seeking out the message tracker. Every proxy that uses the DOLR service becomes a member of it, meaning that Walkabout proxies should ultimately be able to form their own self-sufficient service. However, an initial deployment of Walkabout would only contain a small number of proxies, so they would benefit from the reliability that a larger external service could provide.

Walkabout is designed to cope with sporadic network connectivity, but transfers are unable to take place unless devices have *some* WiFi access. Fortunately WiFi is becoming increasingly widespread. For example, the Wireless Geographic Logging Engine (WiGLE) (WiGLE, 2008b) is a community project that records the information about wireless access points around the world. As figure 4.5 shows, volunteers have collected data on over 13 million access points since the project began in 2001.

People install secured access points in both homes and businesses, so that devices they trust can access the web without the need for network cables. Client devices can therefore use them to access Walkabout proxies too, provided that they know the encryption key. Unsecured private access points are also prevalent, as seen by projects like WiGLE and

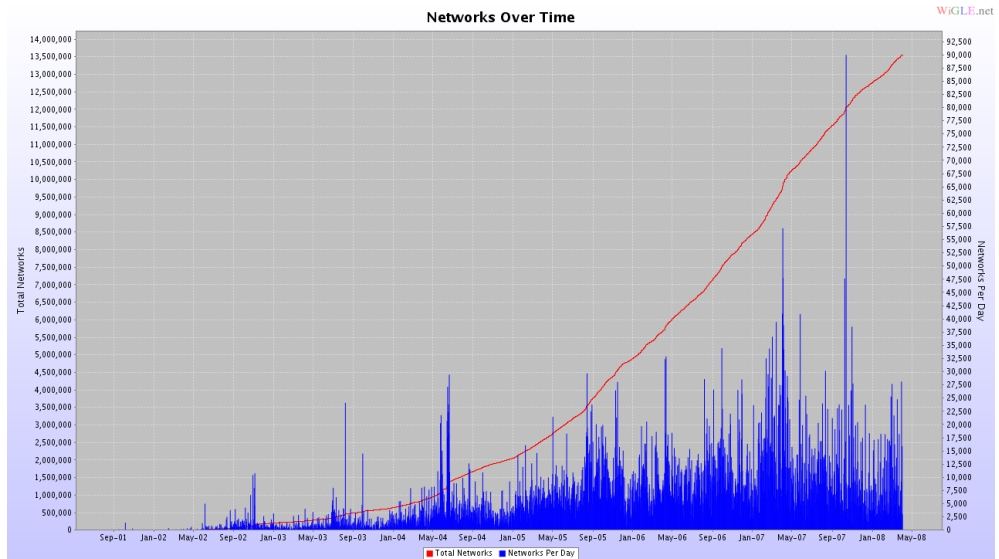
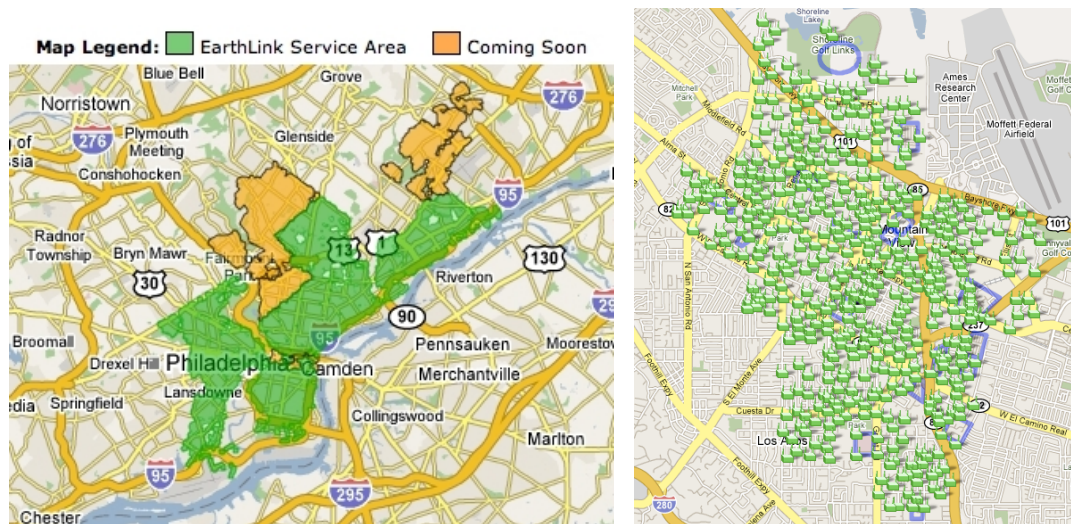


Figure 4.5: A summary of the number of access points recorded by WiGLE (WiGLE, 2008a).



(a) Philadelphia (Wireless Philadelphia, 2007)

(b) Mountain View (Google, 2007d)

Figure 4.6: Coverage maps of the metropolitan WiFi networks in Mountain View, California and Philadelphia, Pennsylvania.

NodeDB (Groth, 2006), but unauthorised use of these networks can be illegal (BBC News, 2007). However, an architecture such as the Collaborative Community Network (Landfeldt *et al.*, 2006) could make it possible for these private access points to legitimately provide a public service, by offering unused Internet bandwidth to foreign devices.

There are also many attempts to provide wide scale public WiFi services in a coordinated manner. These are commonly provided as subscription services, as seen by the HotSpots that companies like T-Mobile (2007) and Boingo Wireless (2007) offer around the world, or the efforts to provide complete WiFi coverage to the cities of London (thecloud.net, 2007) and Philadelphia (Wireless Philadelphia, 2007) (as seen in Figure 4.6(a)). These large areas of WiFi access may also be free at times, although the cost of providing the network means that this is rare. Figure 4.6(b) shows the free coverage that Google has established within their home city of Mountain View, California (Google, 2007d). The government of New South Wales, Australia, also has plans to build free WiFi networks in each of the state's major city centres (Sydney Morning Herald, 2007)

It would be simple to integrate Walkabout with subscription WiFi services. Proxies would be installed alongside (or within) the access points, and client devices would authenticate with them using the secure registration from Section 4.7. If the registration information corresponds to a valid subscription, the service would provide the device with access to upload and download Walkabout data, and bill the user accordingly.

Finally, people need to have devices that are capable of using the architecture. As Section 2.1 demonstrates, there are a wide array of devices on the market today that have WiFi capabilities, ranging from portable computers to PDAs and cameras. WiFi is such a ubiquitous standard that it is built in to most devices, as opposed to cellular networking which usually requires external adapters in anything other than mobile phones. This means that if a user is carrying their device, that can use it to access Walkabout. Bluetooth is another common wireless communication technology, and Chapter 6 presents a system extension that enables it to become a Walkabout access method.

4.9 Prototype implementation

During the course of this research, a working prototype was developed in the Python programming language to test the architecture's viability. This became a valuable tool that helped to refine numerous design details. It is presented here to illustrate what a real Walkabout application would look like and what API a developer would need access to in order to create it.

On the client, the prototype consists of two key elements: the library and the application. The library manages all the client-proxy communications that a device needs to carry out Walkabout transfers, while the application imports the library and dictates when to upload and download data. An application specifies a keypair when it instantiates the client library, to define the identity that the device will take within the architecture. The prototype only permits one application, and hence one identity, per device. A full scale implementation would allow multiple applications per device, with the possibility of each having its own identity or of several applications sharing the same one. The prototype also includes a proxy service, and a basic centralised registration and forwarding service that operates in place of a DOLR.

A backup suite of two applications was also developed as part of the prototype. The backup utility application includes a GUI, so that users can interactively select files from their computer for upload and watch the progress. The storage point application receives these files and automatically saves them to disk. There might be numerous storage points available to a user, so that they can store content to multiple context-dependent locations. This suite is important because it is essentially an implementation of the photo backup scenario from Section 2.2, which is one of the main motivations behind this research.

The following outlines how these applications work. A user runs the backup utility on any device that is able to run Python code and display a GUI. The first time that the user launches the backup utility, he sees an empty version of the main application window presented in Figure 4.7, and clicks the ‘Change’ button to launch the host selection dialogue. As Figure 4.8 shows, he specifies all the storage points he knows and uses, by entering a base 64 encoded string representation of the public key for each one, and a short description of the host they are running on. Once the desired hosts are all saved, he is ready to start the backup. The backup process is as follows:

- The user opens the host selection dialogue, and selects the storage points to use by moving them to the pane on the right. In Figure 4.8, he has selected the ‘Home’ and ‘VerySafe backup server’ locations. He clicks ‘Done’, and control returns to the main window. The list of ‘Current destinations’ at the top of the window reflects these changes.
- He launches a standard file selection dialogue by clicking ‘Add file’, and selects one or more files that he wishes to send to the current destination(s). The application calculates the piece boundaries for each file, generates a header, then adds them to the list of ‘Pending files’ in the top pane. He repeats the process several times to add several different types of files, specifying different storage destinations as desired. In this example, he

4.9. Prototype implementation

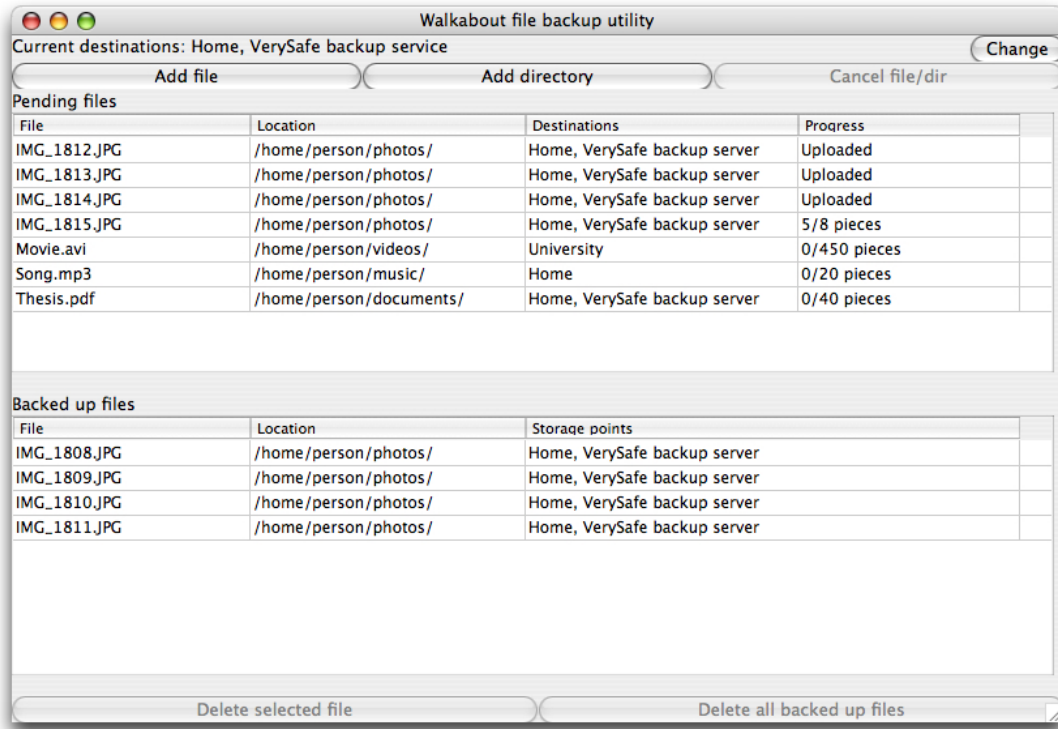


Figure 4.7: The main window of the backup utility, showing the current progress.

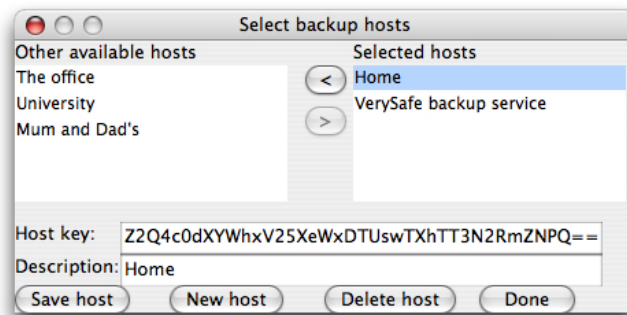


Figure 4.8: The host entry and selection dialogue of the backup utility.

chooses to send photos and documents to both 'Home' and 'VerySafe backup service', videos to 'University' and music to 'Home'.

- When the device has network connectivity, the backup utility automatically locates and connects to the nearest proxy. It uploads the pending files in the order that they appear on the list, by first transferring the header, then any outstanding pieces. The application incrementally updates the on-screen progress indicator for each file as it successfully uploads pieces to the proxy, until it is completely uploaded. Figure 4.7 portrays numerous files at various levels of progress.
- The local proxy initiates the Internet transfer of each new file it receives the header for. It routes the header to the destination proxies, which forward them on to the storage points. Every storage point automatically accepts the transfer, causing its proxy to start downloading the file immediately and forward pieces as they arrive. The storage point writes the data to the same directory structure as the original files, albeit within a dedicated subdirectory.
- The backup utility stops uploading whenever network connectivity is interrupted, but resumes silently upon reconnection to a proxy.
- When the storage point has a complete file, the backup utility receives confirmation of delivery and moves the file from the 'Pending files' pane to the 'Backed up files' pane.
- At any point in time, the user can opt to click the 'Delete all backed up files' button. This deletes the local copies of all files in the 'Backed up files' list, and clears their entry from the window. He may also choose to delete an individual file by selecting it and clicking 'Delete selected file'.

Basic tests demonstrated that this backup suite works in practice. A server within the university ran one proxy, while a home server ran another proxy and the storage point application. A tablet PC running the backup utility was able to successfully store all selected files, even when user mobility caused frequent dropouts in the WiFi connection.

The following sections present the programming interfaces for the library and the application, using Python method syntax. These are quite instructive, because the library API clearly outlines the functions that the core Walkabout design provides to applications, while the application interface shows the main system events that are possible. They are the same interfaces that the implementation of the backup suite uses, which proves they are sufficiently functional to produce Walkabout applications.

4.9.1 Client library API

Initialisation

- **WalkaboutClient**(appObject, configFile)

Create an instance of the client library. `appObject` is the client application, which must follow the interface presented in Section 4.9.2. `configFile` specifies the Walkabout configuration file, containing such information as the location of the user’s keypair and the default download directory.

Uploading

- **sendMsgFromBuffer**(msg, consumerList, pieceLen, appData=None)

Send a message containing the binary data within the buffer `msg` to each of the consumers in `consumerList`, with the piece length specified by `pieceLen`. The optional `appData` attaches an additional string to the message header ¹.

- **sendMsgFromFile**(name, consumerList, pieceLen, appData=None)

Send a message, containing arbitrary binary data drawn from the local file named `name`, to each of the consumers in `consumerList`.

- **sendShortMsg**(msg, consumerList)

Send a short message containing the binary data within the buffer `msg` to each of the consumers in `consumerList`. A short message has less delivery guarantees, but is likely to experience faster delivery times (see Section 4.5).

Downloading

- **downloadMsg**(sig, wantsMsg=True, saveAsFile=True, name=None)

Specify with `wantsMsg` whether the message identified by `sig` will be downloaded. `saveAsFile` specifies whether to store the message in the file at `name`, or to deliver the message data directly to the application. If `saveAsFile` is true but `name` is not specified, then the file name will either be the default value provided in the header (if the original upload was a file) or the hexadecimal string representation of `sig` (if the original upload was a buffer).

¹In Python, a method parameter that is followed by “=<value>” is optional, and the parameter will take that default value if nothing is specified.

4.9.2 Client application interface

Uploading

- **uploadProgressAlert** (*header*, *numPieces*)
Called each time a message piece is uploaded to a proxy. Indicates that *numPieces* of the message with *header* have been uploaded in total.
- **completionAlert** (*header*, *results*)
Called when all consumers have either downloaded or rejected the message with *header*. *results* contains the result for each consumer.
- **failureAlert** (*sig*, *consumer*)
Called if the delivery of the short message identified by *sig* to *consumer* failed, most likely because *consumer* was unavailable when the message was sent.

Downloading

- **wantsMsgAlert** (*header*)
Called when a new *header* arrives. The application responds by calling the library's `downloadMsg` method. This method is not called for short messages, which are downloaded automatically.
- **downloadProgressAlert** (*header*, *numPieces*)
Called each time a message piece is downloaded from a proxy. Indicates that *numPieces* of the message with *header* have been downloaded in total.
- **bufferAlert** (*header*, *msg*)
Called when the message with *header* has downloaded completely, and is provided in the buffer *msg*.
- **fileAlert** (*header*, *name*)
Called when the message with *header* has downloaded completely, and is available at the file *name*.
- **shortAlert** (*msg*)
Called when the short message in *msg* has been received.

4.10 Summary

This chapter forms the first part of Contribution **C2** from Section 1.4. It presents the design of the Walkabout architecture, which provides a store-and-forward delivery mechanism to client devices. The location independent addressing, resumable transfers and message caching within the overlay all work to ensure that transfers will succeed regardless of the mobility patterns of the communication endpoints. The architecture is fault tolerant, secure, and ready for deployment in existing networks. This chapter also details the design and operation of an application that was built upon a prototype implementation of the architecture. Finally, it presents the core client API that applications use to access the message delivery service.

The next chapter tests the performance of the message delivery through simulation, and proves that this design makes it practical for mobile devices to carry out Internet transfers under most circumstances.

Chapter 5

Core system evaluation

This chapter covers many experiments that test the performance of Walkabout through simulation, to determine the circumstances that enable practical transfers involving mobile devices. It starts by exploring some of the inner workings of Walkabout, to outline how different parameters can affect performance. Then it compares the performance of Walkabout to common existing transfer techniques, for a variety of different device mobility times, mobility patterns and network speeds.

5.1 Experimental setup

A series of experiments were designed to test the Walkabout model under a variety of conditions, and constructed using the OMNeT++ / INET framework (Varga, 2001). All network communications used TCP/IP, which OMNeT++ models realistically down to the data link layer. Only the core Walkabout model from Section 4.3 was tested, omitting such features as cache management and security. This section presents the standard test network used in all the simulations, and the metrics used to evaluate the results.

5.1.1 Test network

All of the experiments in this chapter used a test network that represents a simplified view of the Internet, as seen in Figure 5.1. The model ignores the complexity within the core of the Internet, and simply treats it as a cloud with sufficiently high bandwidth to support all transfers at their maximum speed. There are 100 sub-networks connected directly to the Internet cloud, with bandwidth of 100KB/s in both directions and 10ms latency. These symmetric links provide a simple platform to observe Walkabout's performance, but are not necessarily

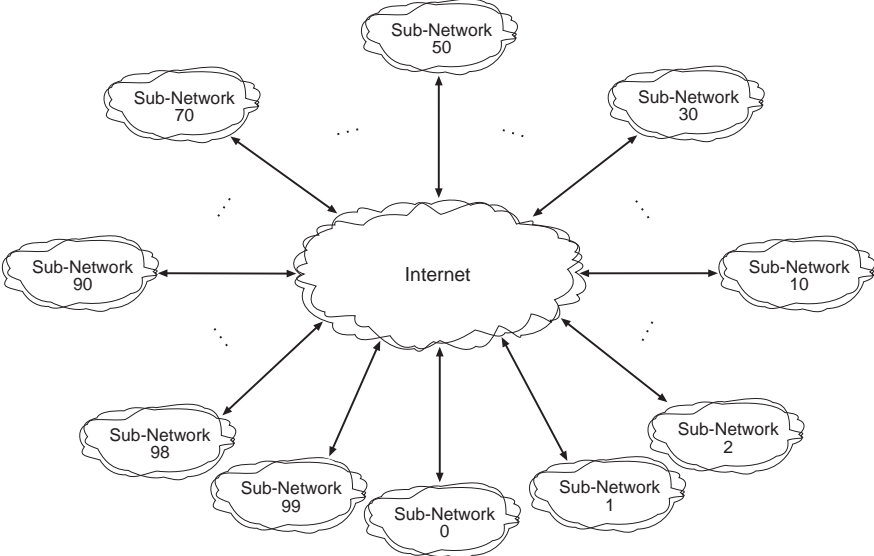


Figure 5.1: The test network.

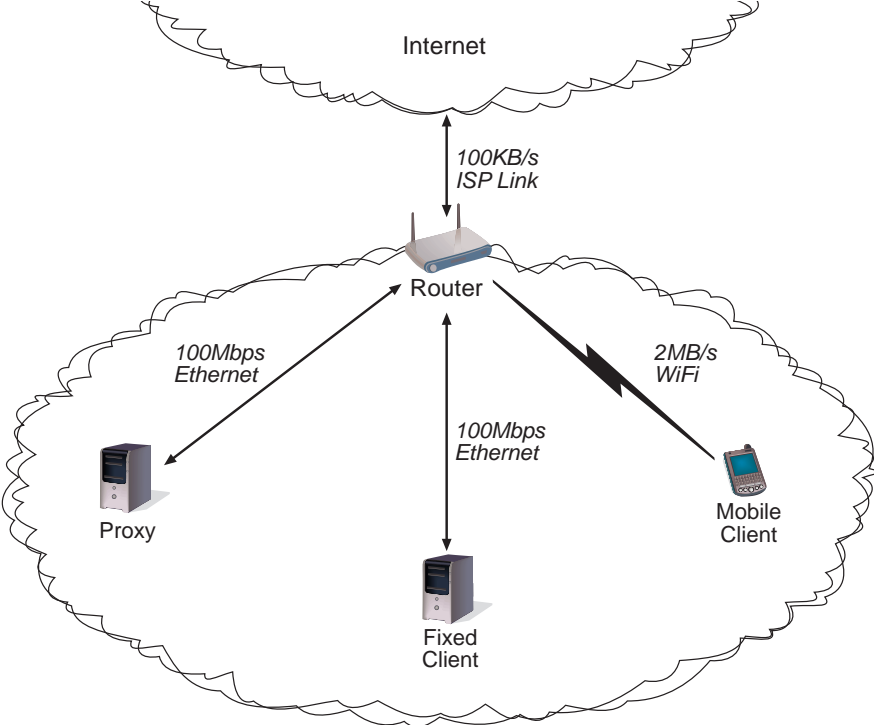


Figure 5.2: The contents of a single sub-network.

indicative of real world conditions. In situations where performance would be affected, the experiments were repeated with the downlink speed from the Internet to the sub-network raised to 500KB/s, to reflect the asymmetric nature common to many home Internet connections.

Each sub-network contains a router with a fixed client and a Walkabout proxy attached via 100Mbps Ethernet links. Figure 5.2 demonstrates what these sub-networks look like. Transfers across the Internet experience a latency proportional to the numerical “distance” between the routers. Communication between networks numbered consecutively, such as networks 10 and 11 or networks 99 and 0, are subject to the shortest latency of 2ms, scaling up to those networks furthest apart, such as networks 0 and 50, which experience 100ms latency.

There are also 50 mobile clients moving around the network, which follow a mobility pattern determined by the particular experiment. Simulating this many clients provides a degree of variation across the different transfers, which operate between uniformly random pairs of clients. Only one transfer operates in the network at any given time unless otherwise stated. When a consumer connects to a router in one of the sub-networks, it does so over a 2MB/s wireless link with 3ms latency (approximating a good quality 802.11g connection). The wireless model is simplified to isolate the mechanisms of Walkabout from the effects of interference and signal strength, resulting in a connection that always provides a constant transfer speed.

Traffic sent over the DOLR service is managed by a virtual registry module, which monitors publications and maps identifiers to proxy addresses. Rhea *et al.* (2004) found that their Bamboo DHT had a mean lookup latency of around 1.5s in the worst case. Therefore, any notifications routed to a DOLR identifier in these experiments experience additional latency as they pass through the central Internet cloud, determined by a triangular distribution on [0.5s-2.5s] with mode 1.5s, before being forwarded on to the appropriate proxy. This is considered to be a pessimistic, yet realistic routing cost.

5.1.2 Evaluation criteria

One of the most important criteria for the evaluation of Walkabout is the *message delivery time*, which is the amount of time from when the producer starts uploading the header for a new message, up until the consumer receives the final piece. Message size varies within the experiments, so a more effective comparison metric is the size of the message divided by the message delivery time, which shall be defined as the *effective transfer speed*. This value takes all delays relating to startup time and data overheads in to account, such that if two transfers have an identical throughput speed, but one of them has more overheads and therefore takes longer to complete, it will have a lower effective transfer speed.

The *total transfer data* refers to the amount of Internet traffic generated over the duration of a transfer that is message related. This figure includes the actual message traffic and, in the case of Walkabout, such control notifications as message headers, peer updates and block requests. It ignores any client-related signalling such as location updates. An important related metric is the *data overhead* that the transfer incurs. This is calculated by:

$$\begin{aligned} \text{Data Overhead} &= \frac{\text{Total Transfer Data} - (\text{Message Size} \times \text{Number of Consumers})}{\text{Message Size} \times \text{Number of Consumers}} \\ &= \frac{\text{Total Transfer Data}}{\text{Message Size} \times \text{Number of Consumers}} - 1 \end{aligned}$$

The criteria of client upload or download speed are also pertinent in some scenarios, as they give an indication of how effectively the client is using the local link when compared to the maximum speed of 2MB/s. This is calculated by dividing the size of the message by the cumulative time for each connection, counted from when they begin the transfer of the first byte of message-related data, up until they complete the last.

Where appropriate, plots include error bars to denote the standard error. However, due to the uniform topology used in these simulations, many of the results exhibit very little variation between runs. In particular, the reference models of direct, FTP and multicast in Section 5.3 typically have standard error of less than 0.5%. When this is the case, error bars are omitted from the plots to aid readability.

The networks used in these experiments feature uniform link speeds, which is highly unlikely to occur in a real world situation, where there is a vast array of possible Internet Service Providers (ISPs), each with a number of different plans available. The results presented here should not be viewed as absolute measures of performance, but rather as a clear indication of the trends that can be expected when Walkabout is deployed in a real network.

5.2 Exploration of base parameters

As with many network applications, the performance of Walkabout is dependent upon the value of several parameters. If set correctly, these parameters can enable the Internet transfer between proxies to take place at speeds approaching their link maximum. If set incorrectly, though, the transfer protocol is subject to overheads and delays that can have a crippling effect on performance. It is therefore very important to find suitable values for these parameters before a fair comparison can be made to other data transfer techniques.

This section explores two of the pivotal parameters, being the number of simultaneous data requests that a proxy makes to each peer and the message piece size. Each parameter is discussed to determine its importance, and the impact that different values can be expected to have upon performance. A range of values for each parameter are then tested through simulation using the network presented in Section 5.1.1, and some general rules are derived.

5.2.1 Request window size

The Walkabout peer-to-peer transfer protocol is receiver-driven, where the proxy downloading data on behalf of a consumer determines the required pieces, then places block requests with the appropriate peers. The number of blocks that a proxy can request simultaneously from each peer is restricted by its *Request Window Size* (RWS). Once this limit is reached, the proxy can not place any further requests until a block delivery arrives from that peer. The size of this window might be expected to have a significant effect on the efficiency of a transfer, so this section explores that.

This window-based transfer scheme is important in making full use of a connection. If a proxy were only able to request a single block at a time, the downlink would remain largely idle during the time it took for a request to reach a peer, and for the first bytes of the new block to arrive in return. Requesting multiple blocks at once keeps the “pipeline” full, such that there is always data downloading while a new request is being serviced, and the link is used to capacity.

The amount of data D_1 that could potentially be downloaded during the time it takes for the response to a block request to arrive is given by the bandwidth \times delay product:

$$D_1 = \text{Downlink Speed} \times \text{Round Trip Time}$$

There are 16 blocks per piece, so the amount of data D_2 that fills a full window of block requests is given by:

$$D_2 = RWS \times \frac{\text{Piece Size}}{16}$$

To ensure that the RWS is large enough to keep the link active at all time, it must have a value such that $D_2 \geq D_1$. The RWS must also be at least 2, so that there will always be at least one block downloading while another is being requested. Therefore:

$$RWS \geq \max\left(\frac{16 \times \text{Downlink Speed} \times \text{Round Trip Time}}{\text{Piece Size}}, 2\right) \quad (5.1)$$

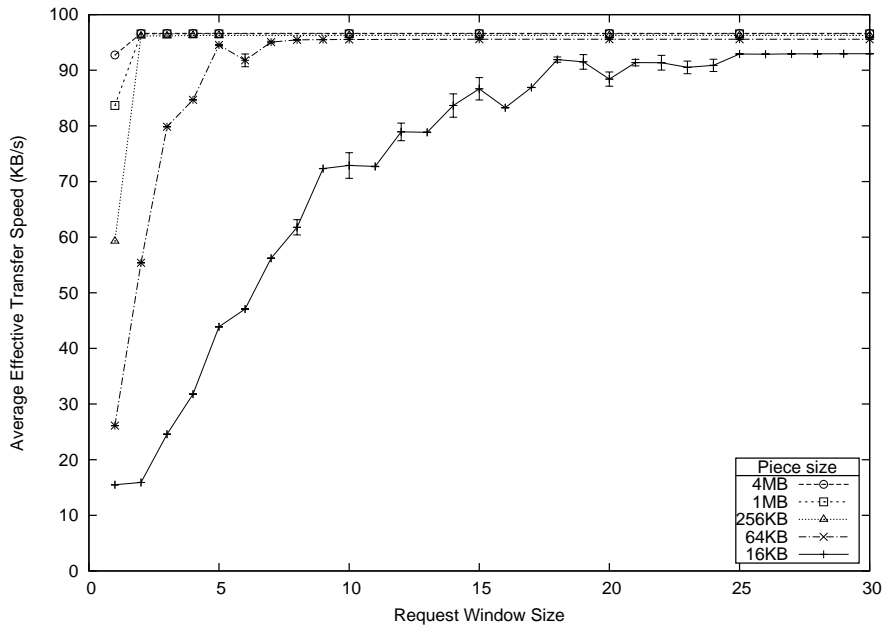


Figure 5.3: Impact of RWS on the average effective transfer speed.

This indicates that the minimum RWS is dependent upon the message piece size and the network properties.

In order to test Equation 5.1, simulations were performed using the network from Section 5.1.1, where each proxy has a symmetric 100KB/s Internet link, and communications with peers are subject to a round-trip-time of between 40ms and 240ms (due to a link latency of between 20ms and 120ms in each direction, dependent upon distance). Message transfers of 50MB to 200MB were carried out between two random fixed clients, with the piece size varying between 16KB and 4MB.

The results are presented in Figure 5.3. As expected, they show that for all piece sizes, an increase in the RWS leads to an improvement in the average effective transfer speed for the file, until the maximum for the piece size is reached. Smaller piece sizes, and therefore smaller block sizes, require a larger window to reach full speed, of 18 for 16KB pieces and 5 for 64KB pieces. They exhibit a large amount of variation, and do not stabilise immediately upon reaching the maximum value, which is an indication of how the susceptible the small blocks are to the changing network conditions between runs. By contrast, the larger piece sizes only require a window size of 2. The RWS values that provide maximum link utilisation for all piece sizes match the calculations for a round-trip-time of under 200ms. If the downlink was faster or the link latency greater, Equation 5.1 indicates that the required RWS would be expected to increase.

While the minimum value for the RWS has a large bearing on performance, the maximum value is not always as important. Each additional request increases the delay between when the block is requested and when it is received, which commits a proxy to making that download further in advance. This is not a problem when there is only a single fixed consumer, as there is only a single source for each block, and they need to get them all eventually anyway. Similarly, when the producer or consumer are mobile, there may be multiple piece sources present across the overlay, but their availability remains largely fixed. Thus the proxy is able to make a block request according to the rules from Section 4.3.4, confident that it will still be the best choice by the time the block is delivered.

When there are multiple consumers, though, the piece availability across the overlay is highly dynamic, and peers need to coordinate their downloads to make the overall transfer as efficient as possible. In particular, peers want to minimise the amount of stress on the *source proxy* (the one that the producer is uploading to), by requesting unique blocks from it whenever possible. Block requests made further in advance, as a result of a larger RWS, are at risk of being poor choices by the time they are serviced. These inefficient block choices would be expected to have an adverse impact on transfer speeds, and Figure 5.4 supports this claim. The figure depicts the relationship between RWS and effective transfer speed, for a range of consumer counts and piece sizes. It shows that there is clear downward trend in effective transfer speeds for larger pieces sizes as RWS increases. Smaller piece sizes are affected less, and actually maintain a steady speed for a wide range of window sizes, as the regular updates between peers and short block delivery times enable efficient piece selection.

More consumers means more peers, and the downlink bandwidth is split between connections to all of them. This will generally lead to a decreased download speed per peer, unless the downlink is fast enough to accommodate the full uplink speed of each peers. According to Equation 5.1, this decrease in bandwidth should lead to a proportional decrease in the RWS required to reach full speed, and this is observable in the 16KB and 64KB plots of Figure 5.4. Similarly, the negative effect that a high RWS has on transfer speed becomes more pronounced as the number of consumers increases in the 256KB, 1MB and 4MB plots, and there is even a slight downturn for 10 consumers in the 64KB plot.

Despite the effects on performance at the lower and upper bounds of window size, each piece size exhibits a value for the RWS where performance stabilises at or near peak performance across all consumer counts. Therefore, the values presented in Table 5.1 will be used for the RWS in all future experiments that use the sample network with 100KB/s

5.2. Exploration of base parameters

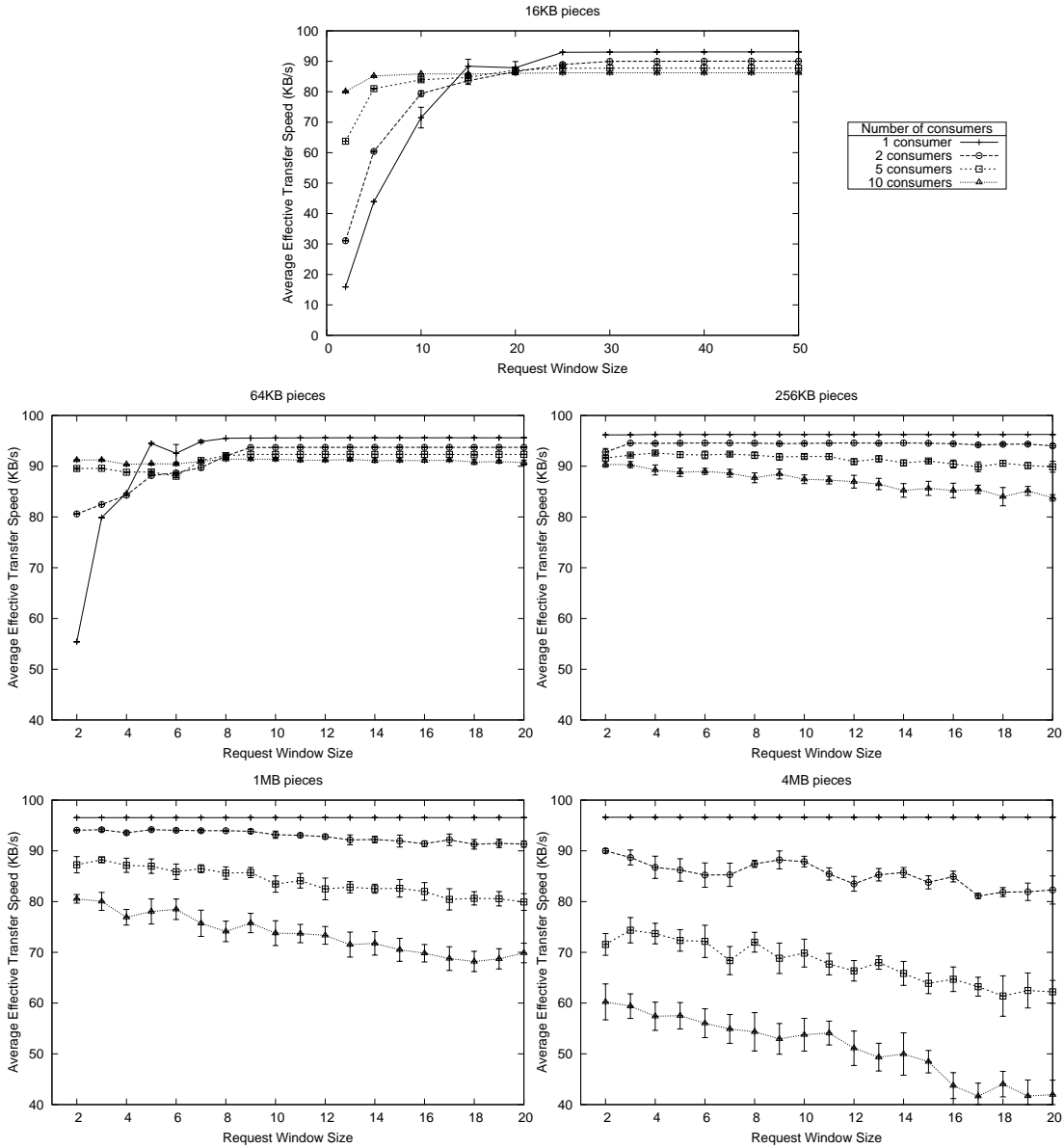


Figure 5.4: Impact of RWS and the number of consumers on the average effective transfer speed.

Table 5.1: Optimal RWS for future experiments.

Piece Size	RWS
16KB	30
64KB	10
256KB	2
1MB	2
4MB	2

connections. However, the RWS is heavily dependent upon the network conditions, so it should not remain fixed in practice. Rather, the calculated window size from Equation 5.1 should be used as a starting point, then adjusted and monitored to achieve maximum performance. It should then be recalculated periodically, so that it adapts to changing network conditions.

5.2.2 Piece size

When a producer first generates a message, it starts by splitting the data in to pieces. The piece size is fixed for the lifetime of the message, and while it may take any value, it should divide evenly by the number of blocks per piece (which is currently set at 16). Piece size has a strong influence on overall transfer efficiency of all the parameters, as this section shows.

The number of pieces in a message directly influences the number and size of most notifications in Walkabout's Internet transfer protocol, and therefore directly influences the amount of excess data generated while transferring the message. It is calculated through the simple relation:

$$\text{Number of Pieces} = \left\lceil \frac{\text{Message Size}}{\text{Piece Size}} \right\rceil \quad (5.2)$$

A series of experiments were carried out to determine how much varying the piece size (and therefore the number of pieces) affects transfers between random pairs of fixed producers and consumers in the network from Section 5.1.1. Transfers of file sizes ranging from 1MB to 1GB were tested against piece sizes of 16KB, 64KB, 256KB, 1MB and 4MB. Figure 5.5 shows the inverse relationship between piece size and average overheads, where a small piece size of 16KB leads to large overheads of 7.25% on average, ranging through to 4MB pieces yielding small overheads of only 0.03%. These differences in overhead translate to the clear differences in effective transfer speed seen in Figure 5.6. The rates for 256KB, 1MB and 4MB pieces are all similar, at approximately 95KB/s for most file sizes, with the difference between the 1MB and 4MB rates indicating that there would be little advantage to increasing piece size beyond 4MB. The slower effective speeds for smaller files are due to the

5.2. Exploration of base parameters

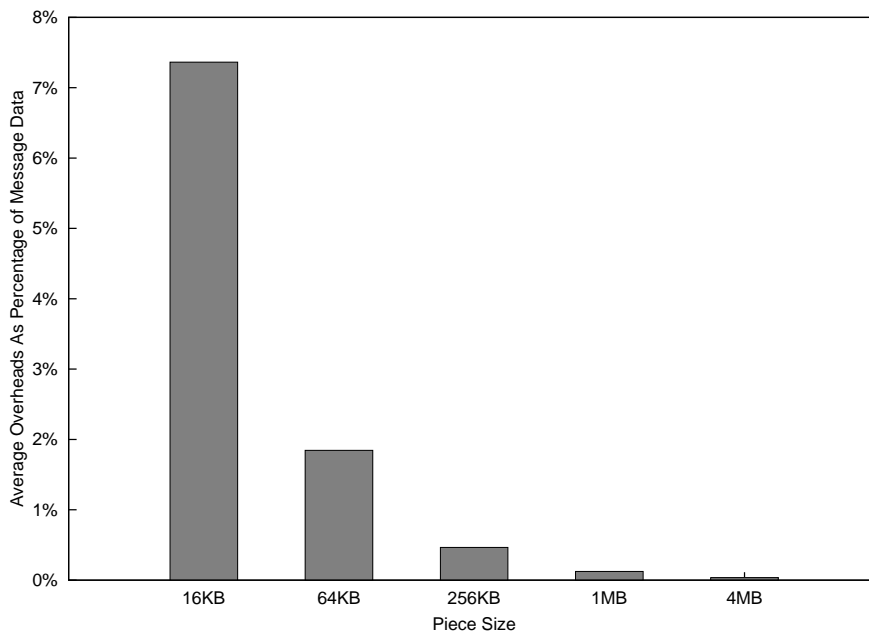


Figure 5.5: Impact of piece size on average data overheads.

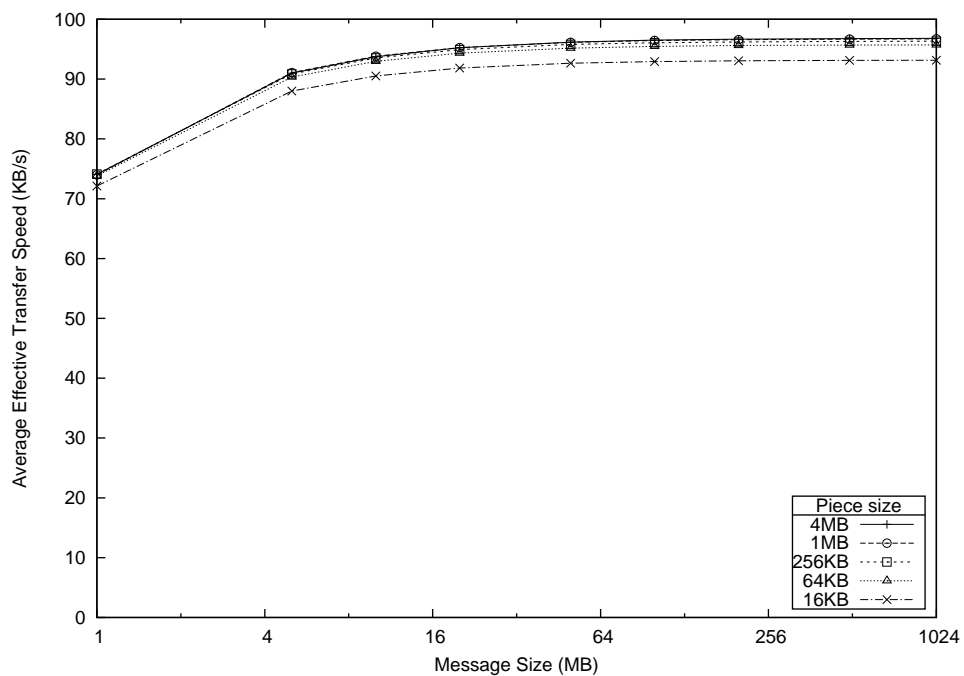


Figure 5.6: Impact of message size on the average effective transfer speed.

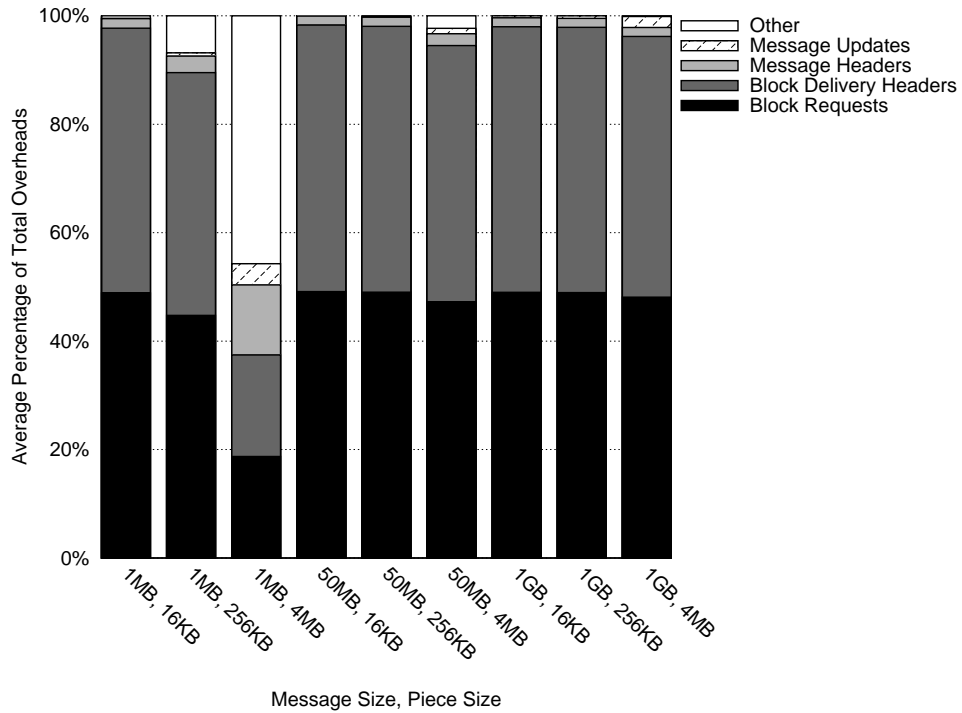


Figure 5.7: Breakdown of the contribution each notification type makes to overheads.

extra time taken to locate the consumer via DOLR and initialise the transfer, where the few seconds of startup cost translate to a significant proportion of the otherwise short transfers.

A breakdown of how much the different notification types contribute to the overheads is shown in Figure 5.7. Block requests and block delivery headers dominate overheads, regardless of piece size. This is because even though there are less blocks to request when the piece size is higher, the header and message updates are proportionally smaller. The unusual case of 1MB transfers with 4MB pieces are the exception because the entire message fits inside one piece, and therefore very few block requests are required to complete the transfer.

These results would seem to indicate that a large piece size is usually the best option, particularly when dealing with the large files that are the target domain for Walkabout. However, this is not always the case. Because a piece is the atomic unit of transfer between a proxy and a client, interruption to a mobile client's network connection will require any incomplete piece transfers to be restarted from the beginning. While the larger piece size has been shown to be effective for fixed clients, it could lead to a significant amount of wasted transfer time if either a producer or consumer is moving. Similarly, a proxy needs more time to download all the blocks across the Internet that are required to complete the larger pieces. Thus the amount of data that they can acquire and transfer successfully to the consumer while it is connected is going to be less than if the piece size were smaller.

Figure 5.8 shows how the different piece sizes affect the transfer speeds when messages are addressed to multiple consumers. Piece sizes from 16KB to 256KB are seen to scale well as the consumer count increases, with only a slight drop in speed. However, there is a steady drop in speed for the 1MB and 4MB piece transfers, even though they are configured with the optimal RWS of 2. Similar to the effects of an increase in RWS seen in the previous section, these drops are related to inefficient block selection. When a peer tries to download a block from the source peer, it selects one from a piece that is currently unavailable on any other peer, to ensure that the source proxy's bandwidth is not wasted uploading the same blocks multiple times. Unfortunately, the proxies do not have any knowledge about what blocks other peers are requesting from the source, and when there are less pieces in the message (as a result of larger pieces) and more peers downloading the message (as a result of more consumers), there is a higher probability that they request the same thing. They do not learn about this clash until one of the peers finishes downloading all 16 blocks that make up the piece, so when piece sizes are large, this can lead to a significant waste of the source's uplink bandwidth.

The relation between the stress placed on the source peer and decreased transfer speeds can be seen quite clearly in Figure 5.9, which shows how different conditions influence the amount of data that the source peer needs to provide. At a minimum, the source must upload 100% of the message, and anything beyond this will slow the overall transfer down. The figure illustrates how smaller piece sizes place a relatively light load on the source, due to there being minimal chance of peers requesting blocks from the same piece, and a relatively small cost when they do. The larger pieces of 1MB and 4MB place more strain on the source peer as the number of consumers grow, which translates to lower effective transfer speeds. In fact, the correlation between data uploaded by the source and the download speed is so strong that Figure 5.8 and Figure 5.9 are almost exact reflections of each other.

Download speeds for multiple consumers could be improved if proxies were to negotiate with their peers before they request blocks from the source. This could significantly reduce the number of duplicate uploads by the source and provide a corresponding increase in effective transfer speed, particularly when piece sizes are large. However, this would also complicate the transfer protocol, and is only suggested here as a future possibility.

5.2.3 Recommendations

Taking all factors in to account, 256KB is the best all-round piece size for single consumer messages. It has very low overhead, which leads to an effective transfer speed only marginally less than the larger piece sizes, and pieces are quick to transfer across local networks

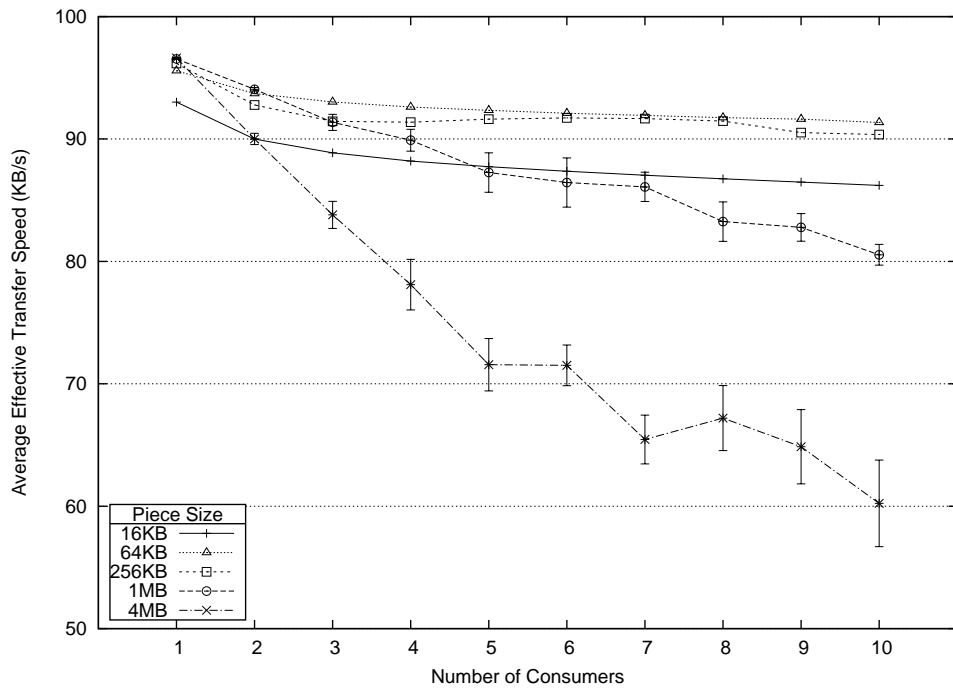


Figure 5.8: Impact of the number of consumers on the average effective transfer speed.

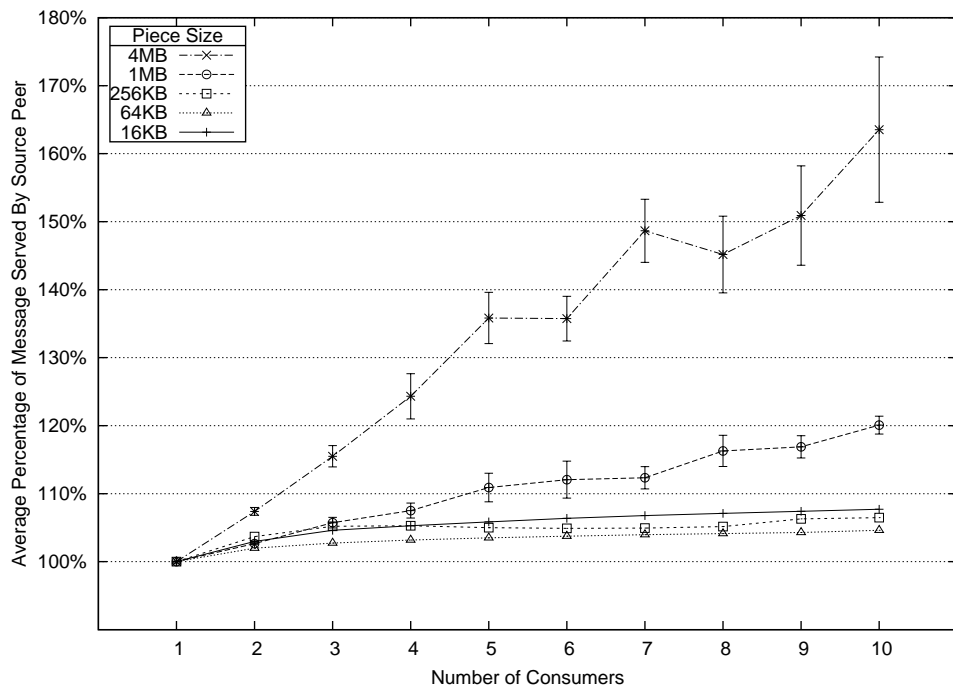


Figure 5.9: Impact of the number of consumers on the source proxy stress.

(about $\frac{1}{8}$ of a second each across a 2MB/s wireless link). It also scales well for messages sent to multiple consumers, with an acceptable decrease in transfer speed. When sending to multiple consumers, though, 64KB pieces present a faster option, as the higher overheads are offset by a more stable, scaleable transfer speed across a wide range of RWS. The RWS is heavily dependent on the network properties, but for messages sent across the test network, a value of 2 is suitable for 256KB pieces, while a value of 10 is suitable for 64KB pieces.

Performance gains could potentially be achieved by increasing the piece size if the participating devices are mostly stationary, or decreasing it if the Internet speeds are low and/or the devices are moving very rapidly, but it is difficult for the producer to make such predictions about the entire transfer process. Thus 256KB pieces were used as the default option for the other single consumer experiments in this thesis, and 64KB pieces for the multiple consumer experiments.

5.3 Comparison to existing techniques

As Chapter 3 demonstrates, existing techniques for transferring data to mobile devices fall into several different categories. This section presents simulations of direct transfer, application-level multicast and FTP models, which represent the registry-, redirection- and storage-based methods respectively. These were tested under the same conditions as Walkabout, to measure their comparative performances. Each experiment featured runs with random message sizes varying uniformly between 5MB and 200MB, intended to represent a range of common sizes for photos, music, and video files, as indicated by Chapter 2.

The direct model requires both the producer and consumer to be connected simultaneously for the transfer to take place. The producer always knows when and where the consumer is connected, and so it resumes the transfer of any interrupted messages when it can reach the consumer, from the exact point in the byte stream where it left off. Cellular transfer is an alternative direct model that is not simulated in this thesis, due to such reasons as impractical cost and speed, as discussed in Chapter 1. If a simple cellular model was included in the results, it would appear as a relatively constant low speed transfer that is unaffected by client disconnection.

The centralised models of multicast and FTP require a dedicated server to also be part of the network. These models introduce a single host attached directly to the Internet cloud by a link with a very high bandwidth of 10Gbps and 1ms link latency, that clients register with when they connect. The producer uploads its message directly to the server while it is connected, and

the method of forwarding depends upon the model. Multicast data is forwarded as it arrives (or held until the consumer is next available), while FTP data is not forwarded until the entire message is available at the server. As with the direct model, broken transfers are resumed from the point in the message where they were interrupted.

The connection patterns that a client connecting to and disconnecting from networks can experience range between two extremes. At one extreme, a consumer may always reconnect back to the same network, while at the other they might be constantly moving, never visiting the same network twice. These patterns are referred to respectively as *reconnecting* and *migrating* from this point forward. While they are used here to illustrate the bounds for simulation results, a realistic connection pattern would be expected to lie somewhere between these two, and Section 5.3.4 investigates this.

The initial tests in this section ran with fixed producers and consumers, to observe the base performance of Walkabout. Once mobile clients were introduced, the client(s) on the other end of the transfer remained fixed, so that the effects of mobility could be observed clearly. Table 5.3 summarises the experimental parameters used in remainder of this chapter.

5.3.1 Fixed producer and fixed consumer

For this simple experiment, the end-to-end delivery times and application data overheads were measured when both the producer and consumer were fixed devices. 50 simulation runs were performed for each model. While Walkabout is aimed at a mobile scenario, it is expected that it should still exhibit an acceptable level of performance in the fixed case, and the results support this.

Table 5.2 shows the effective transfer speeds and data overheads for each of the models. Being the simplest, the streaming models of direct and multicast exhibit the highest speeds, while FTP is half as fast, because the message must first be delivered to the server in full and then delivered the consumer before the transfer is complete. Similarly, both centralised methods experience 100% overheads because the message data is sent across the Internet twice in the course of the transfer. Walkabout transfers experience a DOLR delay during the initial

Table 5.2: Walkabout transfer performance between two fixed devices.

Transfer Method	Avg. Effective Transfer Speed (KB/s)	Avg. Overheads
Direct	96.8	0.0%
FTP	48.4	100.0%
Multicast	96.8	100.0%
Walkabout	95.9	0.5%

Table 5.3: Experimental parameters for comparison to existing techniques, Chapter 5

Exp.	Total Client Count	Producer Mobility Pattern	Consumer Mobility Pattern	Consumer Count	Downlink Speed (KB/s)	Uplink Speed (KB/s)	Conn. Time (s)	Disconn. Time (s)	Message Size (MB)	Results
1	50	Fixed	Fixed	1	100	100	N/A	N/A	5-200 (random)	Table 5.2
2	50	Fixed	Fixed	1-10 (varying)	100	100	N/A	N/A	5-200 (random)	Figure 5.10, Figure 5.11
3	50	Migrating, reconnecting	Fixed	1	100	100	10-1800 (varying)	60	5-200 (random)	Figure 5.12, Table 5.4
4	50	Migrating, reconnecting	Fixed	1	500	100	10-1800 (varying)	60	5-200 (random)	Figure 5.13
5	50	Fixed	Reconnecting	1	100	100	60	10-1800 (varying)	5-200 (random)	Figure 5.14, Figure 5.15, Figure 5.16
6	50	Fixed	Migrating	1	100	100	60	10-1800 (varying)	5-200 (random)	Figure 5.14, Figure 5.15, Figure 5.16
7	50	Fixed	Migrating	1	500	100	60	10-1800 (varying)	5-200 (random)	Figure 5.17, Figure 5.18, Figure 5.19
8	50	Fixed	Real-world	1	500	100	60	10-1800 (varying)	5-200 (random)	Figure 5.22, Figure 5.23

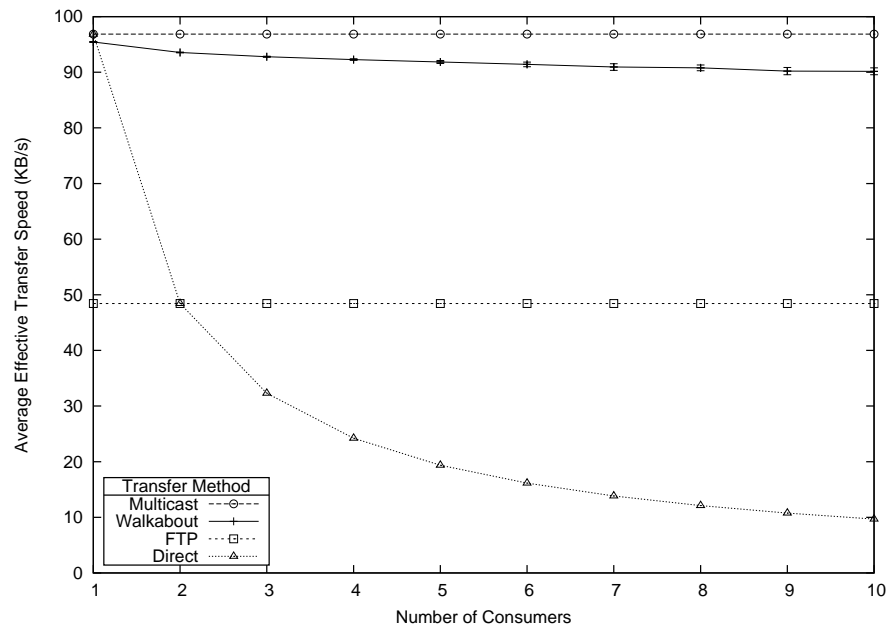


Figure 5.10: Impact of the number of consumers on the average effective transfer speed between a fixed producer and multiple fixed consumers.

transfer establishment, as well as a slight traffic overhead, but once the transfer is established, the pipelining of block requests makes it as nearly efficient as a streaming protocol.

These results confirm that, as a baseline indicator of performance, Walkabout is only 1% slower than a direct transfer when both endpoints are fixed. Therefore if a mobile device were to remain static for an extended period of time, it would not be adversely affected if it were to continue communicating via Walkabout.

5.3.2 Fixed producer and multiple fixed consumers

One of the benefits of a co-operative peer-to-peer transfer system is that it should be expected to scale well for messages sent to multiple recipients, because the proxies that form the overlay can share the load of providing data to each other. Section 5.2.2 confirms this to be true of Walkabout, and Figure 5.10 shows its performance relative to the alternative methods. It strongly outperforms the direct transfer method, which needs to upload an entire copy of the message for each additional consumer and is therefore very badly affected. Walkabout also continues to perform better than FTP, but the massive bandwidth at the server allows the centralised methods to scale to multiple consumers with no change in speed, while Walkabout does trend slightly downwards. The unwavering high speed of multicast suggests that when participants are fixed, its scalability would make it the preferred method to deliver to a large number of consumers.

The decrease in Walkabout's speed as consumers are added is not related to the data overheads of signalling peers, as might be expected. Figure 5.11 confirms that Walkabout experienced only a marginal increase in overheads, from 1.8% for one consumer to 2.3% for ten. Rather, the decrease in speed is due to the proxy block selection algorithm, which causes the source proxy to make inefficient use of its upload capacity as the number of consumers increases, as was explored in Section 5.2.2 and seen in Figure 5.9. Walkabout is not targeted at delivery to a large number of consumers, so this performance is acceptable.

5.3.3 Mobile producer and fixed consumer

This experiment explores the scenario from Section 2.2 of a mobile device uploading to a fixed server. The mobile producer was tested under both reconnecting and migrating connectivity patterns, where the connection time for each run was fixed at a value between 10 seconds and 30 minutes, and the disconnection time was always 1 minute.

Figure 5.12 reveals that effective transfer speeds for the reference models were all badly affected by the shorter connection times, but that Walkabout remained largely unaffected. This is due to the difference between the local network and Internet connection speeds. While connected, a Walkabout producer is able to upload data to its proxy faster than the proxy at the consumer's end is able to retrieve it across Internet. This creates a buffer of message pieces at the local proxy, which continues to provide data for download after the producer disconnects. The buffer is large enough that the transfer is able to continue without interruption until the producer reconnects and starts uploading again. The overall speed is effectively the same as if the transfer was between two fixed devices. The speeds for the direct and multicast models eventually matched those of Walkabout when the session time reached 30 minutes, which was long enough for the entire transfer to take place without interruption under both methods for nearly all file sizes.

Overheads were the same as in Table 5.2, apart from a slight increase in the Walkabout values when a producer was following a migrating connection pattern. This was due to the data cost of introducing a new proxy to the message overlay each time, but was not large enough to affect the transfer speeds. In fact, whether the producer is migrating or reconnecting did not make any difference to the transfer speeds in this experiment.

The buffering is evident when examining the average *producer upload speed* for each method, which is calculated through dividing the message size by the cumulative amount of time that the producer spends connected, between when it delivers the first and last bytes of the message. Table 5.4 shows that this upload speed in this experiment was the same for each of

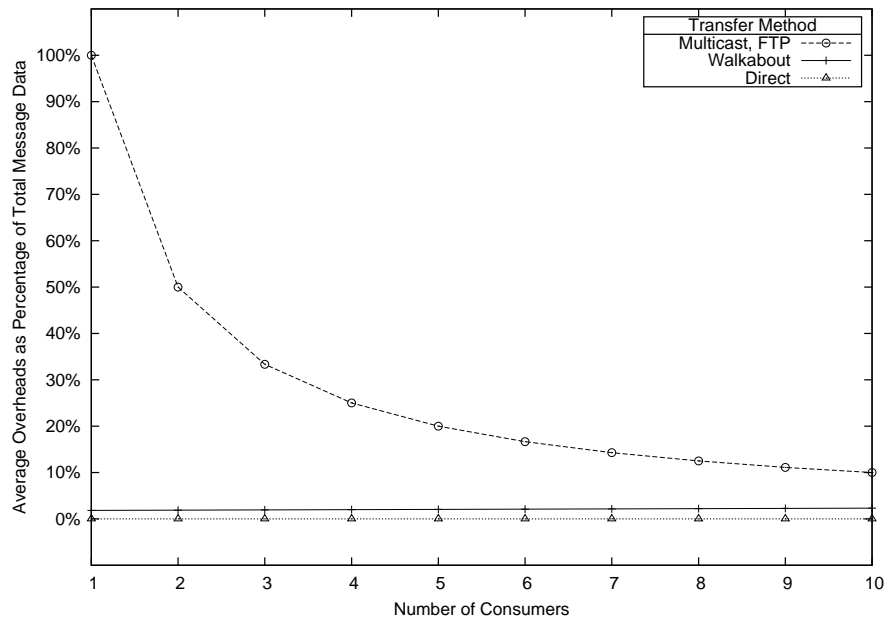


Figure 5.11: Impact of the number of consumers on average data overheads between a fixed producer and multiple fixed consumers.

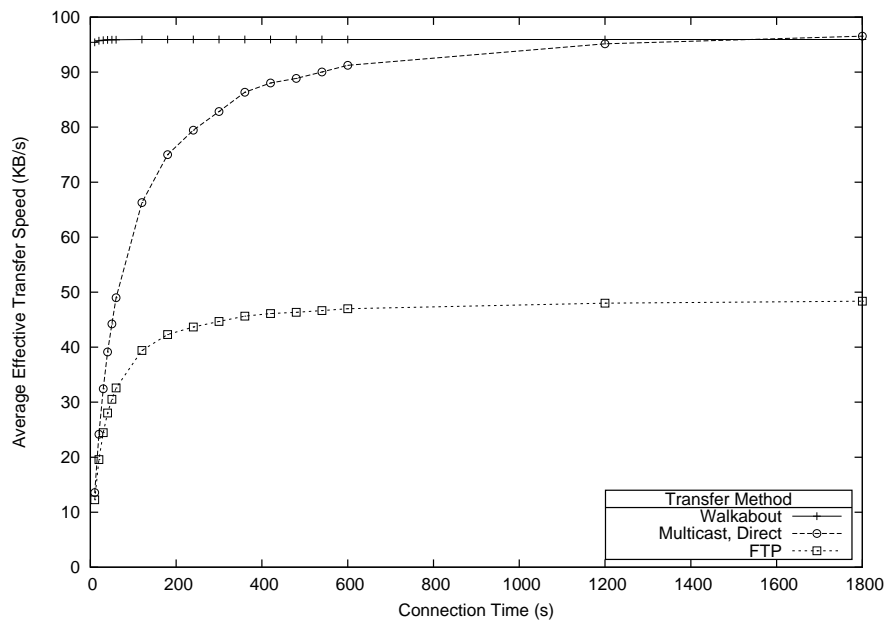


Figure 5.12: Impact of connection time on the average effective transfer speed between a mobile producer and a fixed consumer.

Table 5.4: Average producer upload speed.

Transfer Method	Speed (KB/s)
Direct	96.8
FTP	96.8
Multicast	96.8
Walkabout (10s connection)	1969.8
Walkabout (120s connection)	1981.7

the reference models, at just under 100KB/s, indicating that they were restricted by the Internet uplink speed. By contrast, the average producer upload speed for Walkabout was much faster, approaching the 2MB/s speed of the wireless link. There is a slight setup cost for a Walkabout upload each time a device connects, which explains why upload speeds were marginally lower for short connection times, but they stabilised once a single connection period was sufficient to upload the entire message for all file sizes used in the experiment.

This fast upload speed is one of the key strengths of Walkabout. Mobile devices may only have limited connectivity, but if they can upload their data at local speed, rather than Internet speed, they can inject more of their message pieces into the overlay whenever they are connected. This makes it practical for a mobile user to intentionally linger at a WiFi hotspot long enough to send a large file, or even to continue moving and upload the message in pieces as they come into brief contact with hotspots along their path. Yet despite the small contact time required, the effective transfer speed (and thus end-to-end transfer time) tends to remain better under Walkabout than the alternatives. For example, a Walkabout device with 10 second connection periods in this experiment was able to deliver its message to the fixed consumer faster on average than a direct transfer device with 20 minute connection periods!

As a general rule, a mobile producer moving in and out of network coverage can sustain an uninterrupted full speed transfer if, each time they disconnect, they had already been connected long enough for the proxy to buffer the transfer until they reconnect somewhere. The transfer can continue if:

$$\text{Disconnection Time} \leq \frac{\text{Local Uplink Speed} \times \text{Connection Time}}{\text{Internet Uplink Speed}} \quad (5.3)$$

Note that this is a simplification, omitting such factors as the client-to-proxy session establishment time that slightly reduces the available connection time. This equation holds true when the producer is returning to the same network each time, but the next section explores some additional interesting results that can arise if a consumer is moving between networks where the downlink speeds are greater than the uplink speeds.

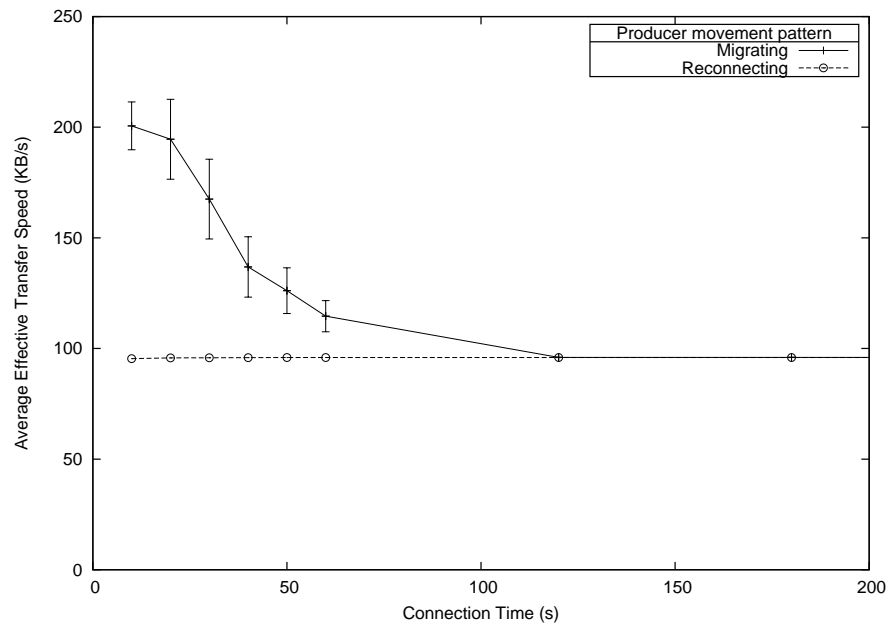


Figure 5.13: Impact of connection time on the average effective transfer speed between a mobile producer and a fixed consumer under Walkabout, when network links are asymmetrical.

Asymmetric networks

Until now, the experiments in this chapter have only considered symmetric links between networks and the Internet, because one-to-one transfers are generally limited by the speed of the slowest link along their connection path. However, interesting results are observable when a producer is moving rapidly between networks with asymmetric links to the Internet, as is common in home Internet connection technologies such as ADSL. To test this, the previous experiment was repeated to compare reconnecting Walkabout producers to migrating ones, with the Internet downlink speeds raised to 500KB/s. The RWS remained at the optimal value of 2, because even though the downlink speed was greater, the uplink speed still limited the individual connections to 100KB/s.

As Figure 5.13 shows, the reconnecting transfer speed remained the same as in the previous experiment, as it was restricted by the 100KB/s upload speed of the producer-side proxy. However, when the producer was moving rapidly between networks, there was a noticeable increase in transfer speed to over 200KB/s. At lower connection times, a migrating producer is able to upload data to multiple proxies before it exhausts its supply of unique pieces, which creates buffers of data in multiple locations, provided that the ratio of each disconnection to connection satisfies Equation 5.3. This provides the proxy downloading on behalf of the consumer with multiple sources, and it is able to download from them in parallel.

This is an interesting result, because it shows that Walkabout can increase the effective transfer speed for mobile devices beyond what is possible for fixed devices in the same situation. By creating multiple repositories for a message, it is effectively increasing the available upload speed, which is often the bottleneck in transfers over asymmetric links. This property could potentially be exploited if the producer wanted to accelerate the transfer, by repeatedly uploading the message to different proxies, to increase the overall availability and therefore the effective transfer speed.

5.3.4 Fixed producer to mobile consumer

This experiment represents the scenario from Section 2.3 of a mobile consumer device downloading a file from a fixed server. When a connection is available, the download to the consumer is simply a direct transfer for all methods, so there is little of interest to observe. However, different disconnection times alter the amount of data that a Walkabout proxy can buffer in a consumer's absence. This can lead to download speed improvements if the consumer returns to the same proxy, but can also increase data overheads if they do not. Therefore, the connection time in this experiment was fixed to 1 minute, while the disconnection time varied between 10 seconds and 30 minutes for each set of runs. All transfers began during the consumer's initial connection period.

As the consumer disconnection time increases, it accounts for an increasingly dominant proportion of the message delivery time. This has an unavoidable impact on the effective transfer speed, such that even the very best methods trend downwards sharply. By contrast, good transfer methods will improve the consumer's download speed while it is connected, by getting data to the correct proxy before it arrives, or providing ways for the proxy to download on demand at higher rates. So when a consumer is mobile, the *consumer download speed* (measured from when the consumer receives the first byte of the message) is a better indication of performance, and is the focus of the following sections.

The performance of direct, FTP and multicast transfers do not change as a result of the consumer's connection patterns, but Walkabout is extremely sensitive to them. Therefore, this experiment used migrating patterns for the reference models and both reconnecting and migrating patterns for Walkabout.

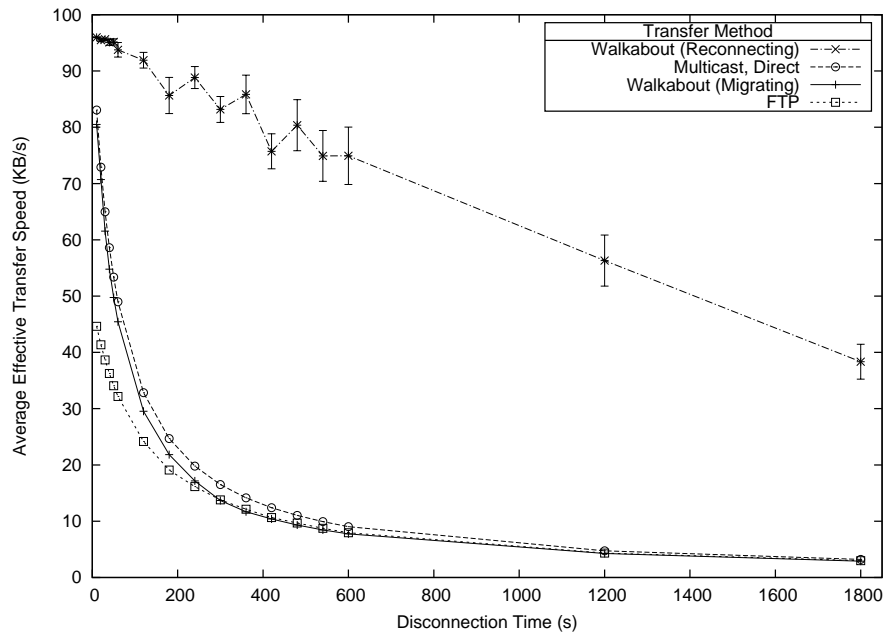


Figure 5.14: Impact of disconnection time on the average effective transfer speed between a fixed producer and a mobile consumer.

Reconnecting consumer

A consumer device could keep returning to the same proxy over the duration of a transfer if it is within a building or on a campus, but disconnect periodically due to fluctuating network coverage or to preserve power. In this localised movement scenario, the consumer-side proxy relays pieces to a connected consumer as it receives them, or buffers them if the consumer is absent. Upon reconnection, the consumer downloads the buffered data at local speeds, which offsets some of the time lost to disconnection. Figure 5.14 shows that this buffering enables Walkabout transfers to have a much higher average effective transfer speed than the alternatives, which are restricted to downloading over the Internet when the consumer is present. As the disconnection time rises, the main reason for Walkabout's decrease in speed is the time lost to disconnection itself, because the proxy completes the download in the consumer's absence, but can't actually deliver it until they reconnect. Even then, the transfer speed remains close to what would be possible if the consumer was fixed instead, for all but the longest disconnection times.

The increased buffering at the proxy as disconnection times increase leads to a larger average consumer download speed. Figure 5.15 shows that while it is significantly faster than the Internet speed, the average never reaches the maximum possible local speed of 2MB/s in this experiment. This is because the consumer downloads at this speed when it first reconnects,

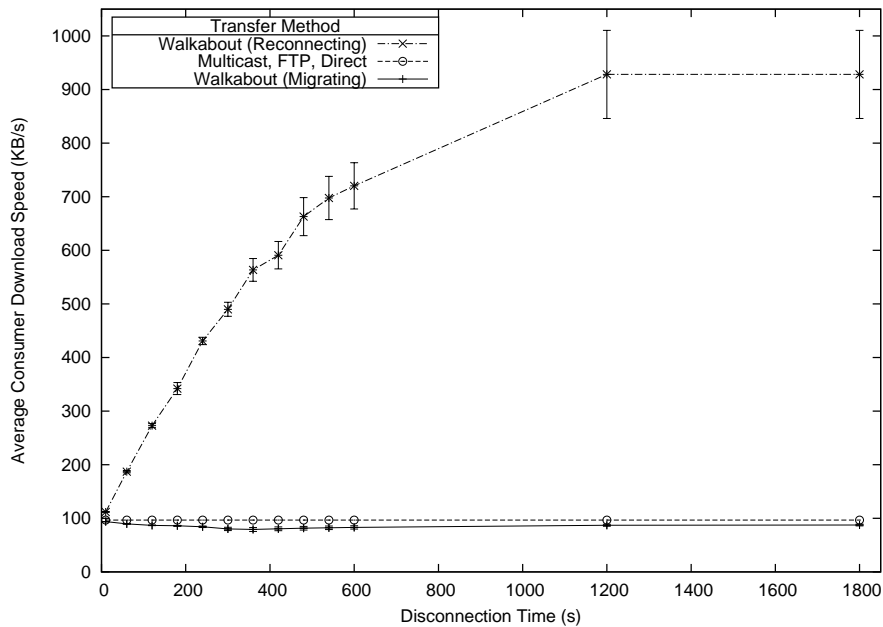


Figure 5.15: Impact of disconnection time on the average consumer download speed for a mobile consumer.

but quickly exhausts all the data in the buffer and falls back to receiving pieces at Internet speed. The download speed for the reference methods are all equivalent, as their bottleneck is the speed at which the proxy can download data on demand.

In a similar fashion to how a faster upload speed is useful to a producer device, this increased download speed can be useful for the consumer device. A user does not need to maintain a connection to download a message, but can instead leave their device off or disconnected from the network (to preserve power, for example) and connect periodically to download the available messages in bursts. It also means that if the user is in an area that has unreliable network coverage, they can carry out large transfers at much the same speed as if they were directly connected, even if their connection is prone to dropouts.

Migrating consumer

In contrast to the reconnecting scenario, Figure 5.14 shows that the effective transfer speeds for a migrating Walkabout consumer are instead comparable to those of the reference models, albeit slightly less on account of the time spent setting up the transfer at each new proxy. This is because there is no chance for the proxies to build up a reserve of message data, and they must resort to downloading it on demand when the consumer connects. This also restricts the consumer download to the speed at which the proxy can stream the data. However, Figure 5.16 reveals that the data overheads increase proportionally with the increase

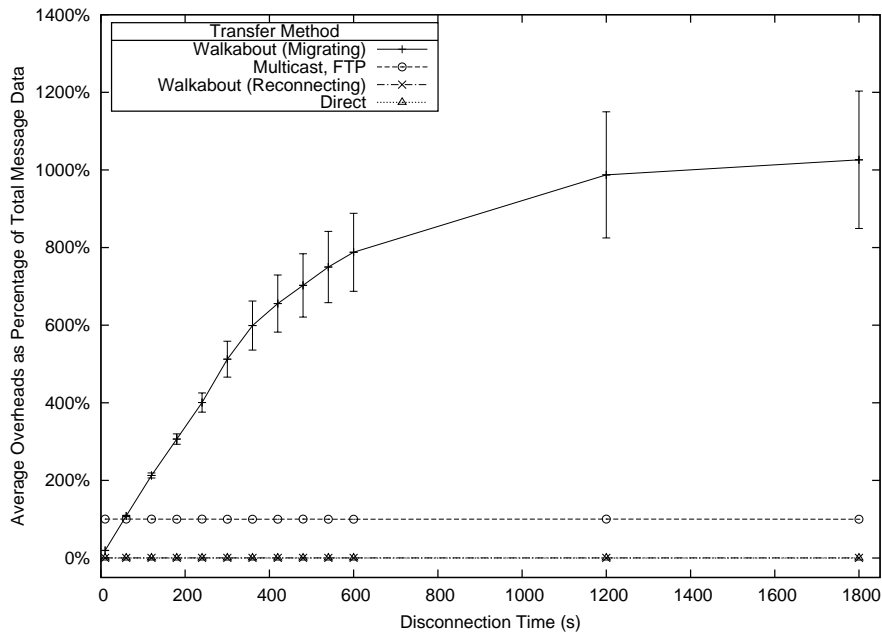


Figure 5.16: Impact of disconnection time on average data overheads between a fixed producer and a mobile consumer.

in disconnection time, climbing to over 1000% when devices disconnect for half an hour. These overheads are the result of the proxies continuing to download blocks during the consumer's absence, such that if the consumer never returns to that proxy, the time spent downloading all these pieces is effectively wasted. It should be noted that the overheads only start to plateau when the disconnection periods are long enough that each proxy runs out of unique data to download.

These overheads are very large, but modifications to the Walkabout protocol could reduce them. The simplest method is for the proxy to only download a message while a consumer is connected, but this eliminates the potential for speed benefits if the consumer does return to that proxy in the future. A better approach would be to selectively choose which consumers to download for. By recording how often it has seen a client, a proxy could decide whether the client is likely to return, and therefore whether it should continue to download pieces in their absence. The ideal approach, though, would be to predict where the consumer is going, and to have that proxy download the pieces instead. The next chapter explores extensions to Walkabout that enable this.

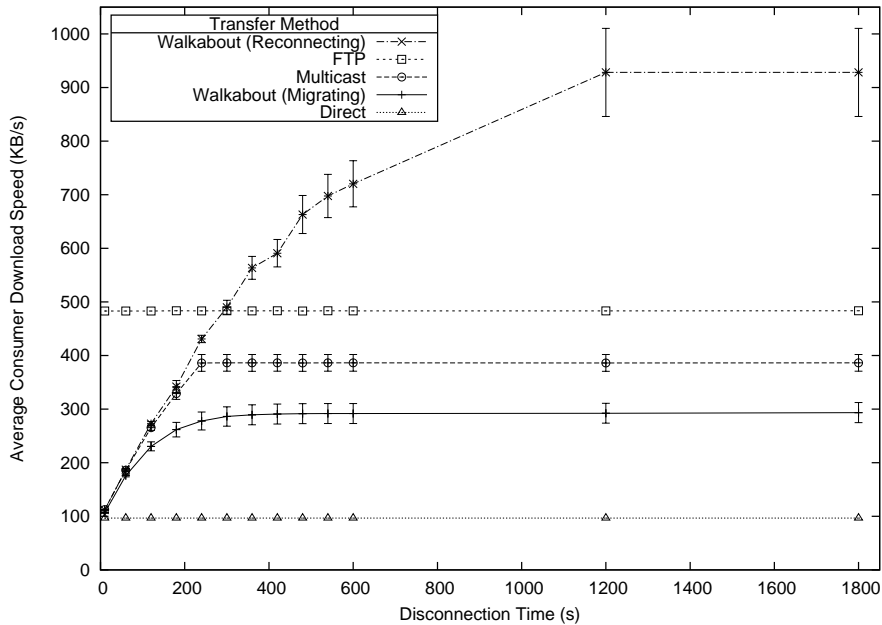


Figure 5.17: Impact of disconnection time on the average consumer download speed between a fixed producer and a mobile consumer, when network links are asymmetrical.

Asymmetric networks

Asymmetric links can improve the effective transfer speeds for migrating consumers. It is not possible to raise them beyond the performance of a reconnecting client, as in the mobile producer scenario, but they can at least get closer to this upper limit.

As evidenced by Figure 5.17, raising the Internet download speeds for each sub-network to 500KB/s then repeating the previous experiment led to consumer download speed improvements for migrating Walkabout, FTP and multicast (when compared to results for the symmetric network in Figure 5.15). The common factor is that when the consumer connects, the new proxy is able to take advantage of the improved downlink speed to acquire data at a faster rate than the producer can provide directly.

This is an obvious consequence for the centralised methods, as the very high bandwidth at the server can saturate the proxy’s downlink until all the stored data is exhausted. For multicast, the length of the intermediate disconnection periods dictate how much stored data there is. Once it is long enough for the producer to upload enough data to sustain a full speed download during the ensuing connection period, the speed levels out. The consumer download speed during the initial connection is only 100KB/s, as there is no stored data on the server, and this keeps the average speed from reaching 500KB/s. By contrast, FTP always supports the maximum consumer download speed, but this is because the consumer-side transfer does

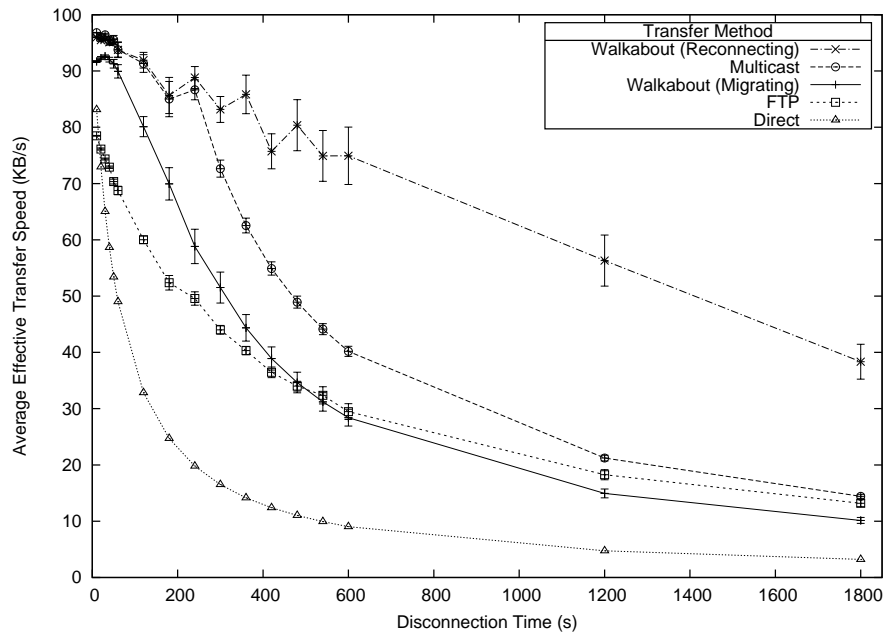


Figure 5.18: Impact of disconnection time on the average effective transfer speed between a fixed producer and a mobile consumer, when network links are asymmetrical.

not begin until the entire message has been uploaded to the server. So the consumer download speed remains fast, but the initial delay means that the the improvement in the effective transfer speed is not quite so dramatic. Figure 5.18 shows that while multicast always has a higher effective transfer speed than migrating Walkabout, FTP does not until the disconnection period grows quite large, when the high consumer download speed offsets the long startup time.

The improvement in migrating Walkabout is a result of parallel transfers. Each proxy that the consumer visits will continue to download new pieces after its departure. Ideally, the consumer would reconnect to download them at local speed, but if it connects elsewhere instead, these pieces become available as an additional download source to the new proxy. When there are multiple peers offering pieces that the consumer needs, the proxy is able to download from them all simultaneously, at a speed limited by either their combined uplink, or its own downlink. This increases the consumer's download speed, thus reducing the amount of time it needs to be connected to complete the message.

Creating these multiple data sources is the key to improving consumer download times under asymmetric network conditions. Larger messages, slower link speeds or shorter consumer connection periods cause a migrating consumer to visit more unique proxies before a transfer completes. However, multiple peers only add to the download speed if they can provide pieces that the consumer actually needs. Longer disconnection periods increase the number of useful pieces that each proxy acquires, enabling them to contribute to accelerated

5.3. Comparison to existing techniques

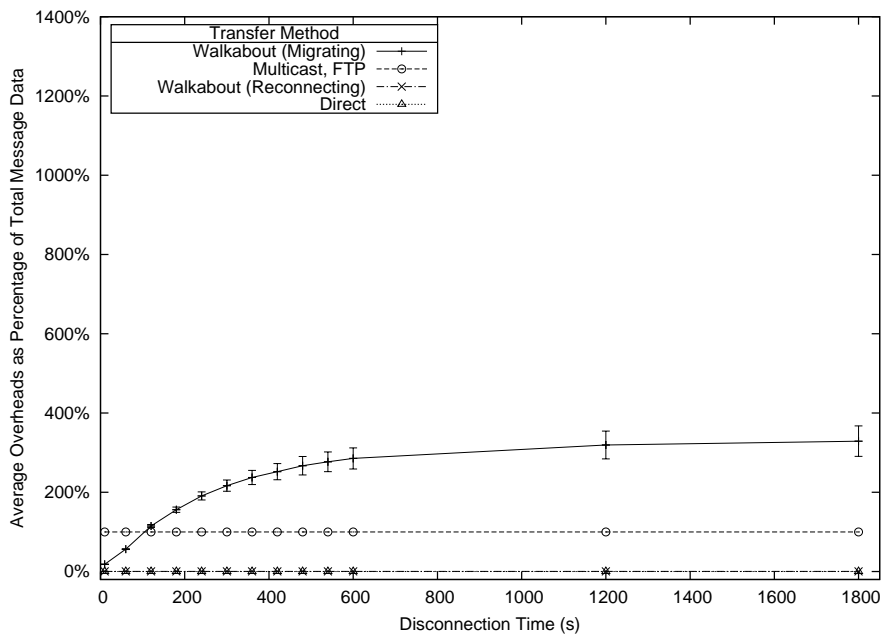


Figure 5.19: Impact of disconnection time on average data overheads between a fixed producer and a mobile consumer, when network links are asymmetrical.

downloads further in to the future. Figure 5.17 shows this relation between disconnection time and a migrating consumer's download time. Even if conditions are favourable, the increase in consumer download speed is a gradual process, which ramps up with each additional proxy the consumer visits. This explains why the download speed shown here never reach the maximum downlink speed of 500KB/s (on average).

These improved speeds also have the benefit of reducing Walkabout's overheads, as seen in Figure 5.19. This is because after a consumer departs, the proxy continues downloading until the consumer rejoins the network, or it has all the pieces that the consumer still requires. Faster download speeds allow a consumers to acquire more pieces per connection period, so these needs grow progressively smaller, and each successive proxy will download less than they would if links were slower. Unfortunately, the overheads still remain quite high, so simply raising the download speed does not remove the need for a predictive solution.

On the whole, the asymmetric connections benefits FTP and multicast more than migrating Walkabout, although it does see some improvement. Neither reconnecting Walkabout or direct transfers benefit from the downlink speed increase, as they remain solely restricted by the uplink speed at the producer.

Real-world movement patterns

As has been stated previously, the reconnecting and migrating patterns are the extremes of possible client movements. In reality, a client would be expected to have a movement pattern that falls somewhere in between, where it visits a selected set of locations on a regular basis, but does occasionally encounter new locations. The previous experiments in this chapter show that Walkabout performs poorly when a consumer is migrating, but do not indicate how common this movement pattern is. Therefore, this section determines what a realistic movement pattern looks like and how a mobile consumer performs when it follows one.

The movement set collected at Dartmouth College, New Hampshire, USA, between 2001 and 2004 (Kotz *et al.*, 2005) contains a large amount of user mobility data. It comprises the logs for 623 access points across 182 buildings, presented as the access point connection history of 13889 unique wireless devices over this period. The size of the campus (approximately one square kilometre – see Figure 5.20) and the numerous devices following regular patterns makes this data set the ideal basis for a realistic Walkabout trace set.

The experimental movement set was created by simplifying these device connection histories. Consecutive connections within the same building were condensed down to a single connection, so that the traces became a series of transitions between buildings. This had the side-effect of removing the possibility that a device could reconnect immediately to the same network it was last at. The processed device traces were sorted by how many connections they made, and the top 50 most populous were selected for use by clients in the experiments. These traces were selected to represent highly mobile devices that can thoroughly test Walkabout's performance. The experiments focused on the real-world sequence of movements, but provided their own connection and disconnection times, so that the results were readily comparable to the other movement patterns.

The underlying movement patterns in the resultant traces are observable in the *path lengths*. Each time a client connects to a network, the path length indicates how many connections it has been since since the client was last there. When a device is migrating, all path lengths are 0 (as it is always visiting a new network), while a reconnecting device always has a path length of 1. In practice, the path lengths would be expected to vary. Figure 5.21(a) shows the distribution of path lengths across the entire movement set, while Figure 5.21(b) shows the results for the top 50 most populous traces, which were the ones used in the experiments. They reveal that the realistic movement patterns tend to be quite repetitive, with the majority of movements being to a network the device has visited recently. In fact, most movements have a path length of 2, where a device has moved from one network to another, then back again.

5.3. Comparison to existing techniques

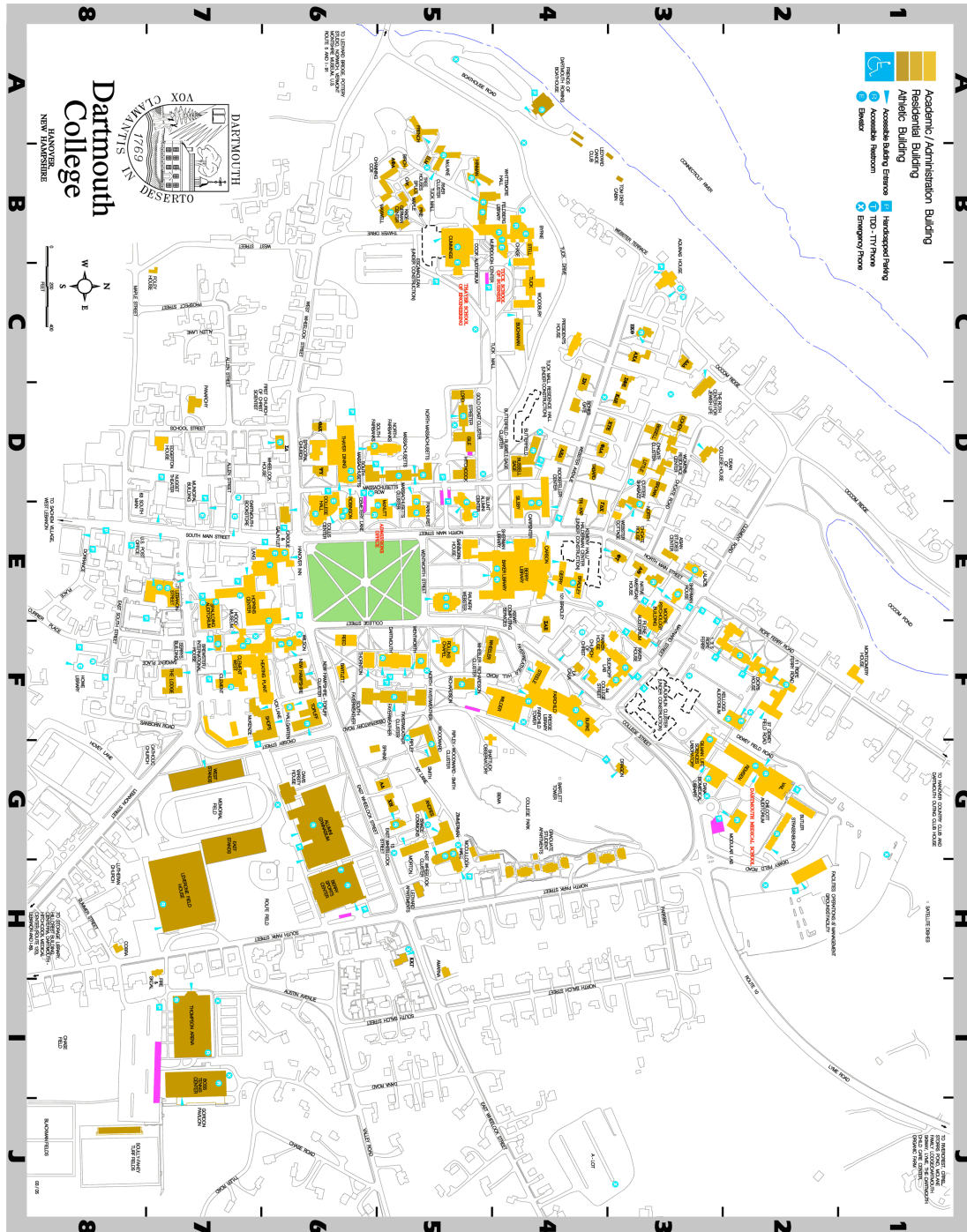


Figure 5.20: The main campus of Dartmouth College (Trustees of Dartmouth College, 2007).

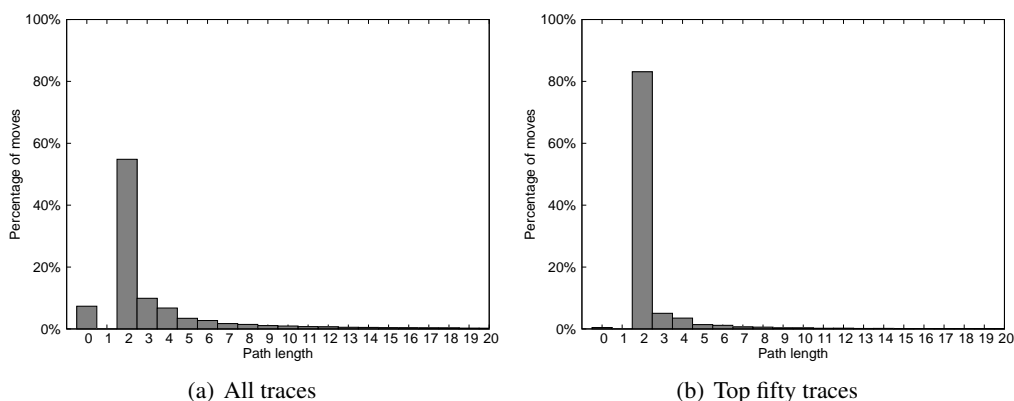


Figure 5.21: Path lengths derived from the Dartmouth college movement traces.

This effect is most pronounced amongst the top 50 traces, where 83% of all movements are of path length 2. This repetitive motion should be expected to suit Walkabout well, as there is a high probability that when a client moves, it will connect to a location that has previously downloaded data on its behalf.

To test how Walkabout performs under these real-world movement patterns, they were applied to a repeat of the experiment from Section 5.3.4. Each of the 182 Dartmouth campus buildings was established as an individual entity in a neighbourhood. Each building had its own sub-network, which was similar to the one in Figure 5.2, except that its Internet link bandwidth was an asymmetric 500KB/s downlink and 100KB/s uplink. The 50 clients moved around the network following patterns derived from the top device traces, with a fixed connection and varying disconnection time.

Figure 5.22 presents the average consumer download speeds under these conditions. The non-Walkabout transfer methods remain unaffected by the change in movement pattern, as the consumer simply downloads directly whenever it is connected. By contrast, the real-world movements do have a large impact on Walkabout, with the results falling between the levels for reconnecting and migrating consumers. The speeds improve upon those for migrating consumers, due to the repetitive movements and the high chance that a consumer will revisit a proxy that has pieces available for it to download at local speed. The different device patterns mean that there is quite a large spread of results, but as disconnection times increase, Walkabout's consumer download speeds are higher on average than the other methods under real-world conditions. They do not reach the speeds that a reconnecting consumer experiences, though, showing that there is still scope for improvement.

5.3. Comparison to existing techniques

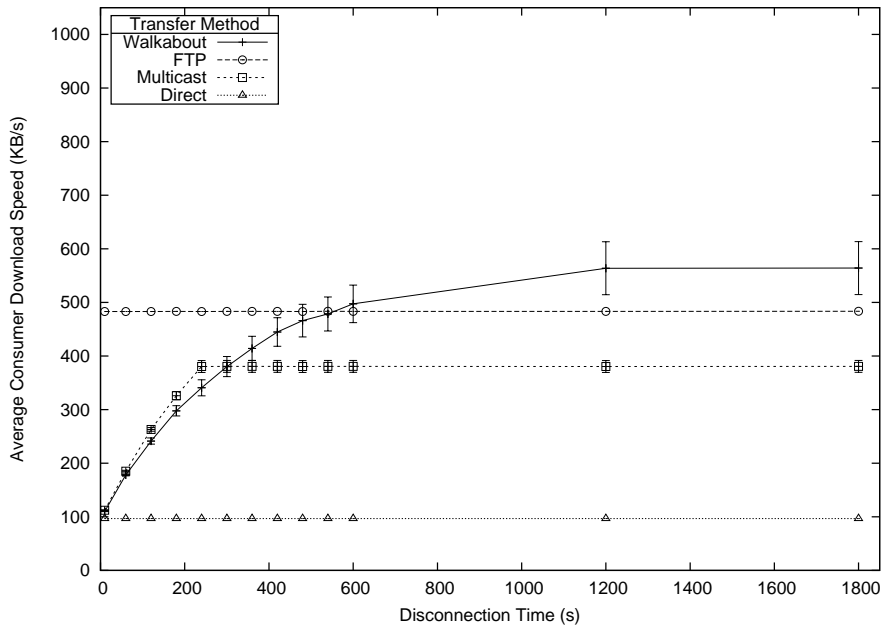


Figure 5.22: Impact of disconnection time on the average consumer download speed between a fixed producer and a mobile consumer following real-world movement patterns.

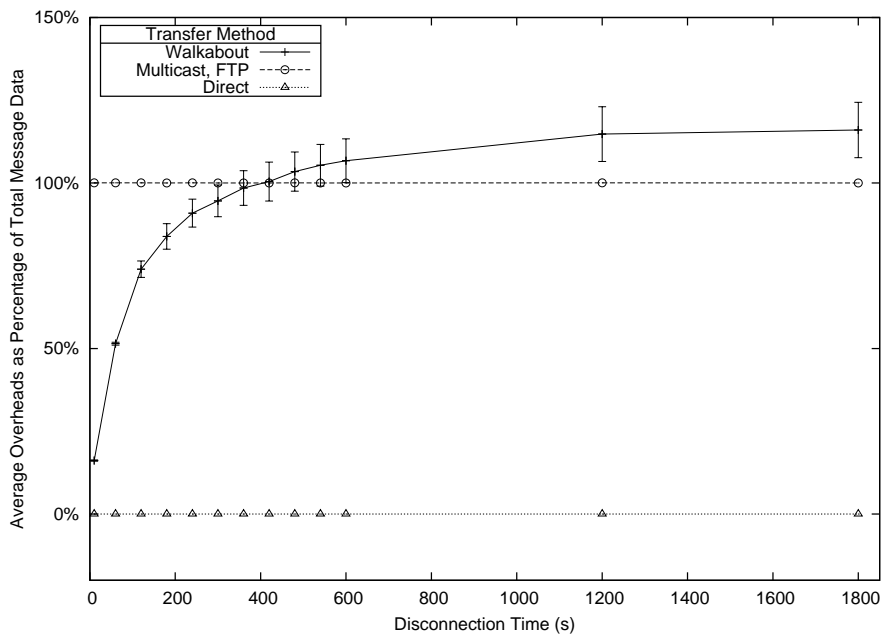


Figure 5.23: Impact of disconnection time on average data overheads between a fixed producer and a mobile consumer following real-world movement patterns.

The Walkabout overheads for a real-world consumer, as seen in Figure 5.23, are also between those for a reconnecting and a migrating consumer. They are significantly lower than the high levels that a migrating consumer experiences, but still rise quickly to over 100%, exceeding the overheads for all of the other methods. If the Internet links are under-utilised, then these overheads might be an acceptable trade-off in pursuit of the potential speed increases that repetitive patterns can produce. While the real-world patterns do improve overheads on the whole, but they are still quite high, and the system extensions in the next chapter aim to reduce them further.

5.4 Summary

This chapter shows that Walkabout scales well to large files and multiple consumers. Transfers achieve high performance levels under any kind of producer mobility, and when a consumer is fixed or constantly reconnecting to the same proxy. These conditions enable effective transfer speeds that are close or equal to what would be possible between fixed devices, and therefore make it practical for mobile devices to carry out large data transfers across the Internet. A rapidly moving producer can also lead to transfers that actually exceed the maximum possible for a fixed device, by delivering pieces to multiple proxies in a way that increases the effective uplink speed that is available to them. Together, these results complete Contribution C2 from Section 1.4.

Walkabout transfers to mobile consumers that are moving between networks remain impractical, as a result of the low speeds and high overheads that they incur. The next chapter formulates an extension to Walkabout that coordinates pre-emptive message delivery, so that consumers following any mobility pattern will find cached pieces waiting for them when they connect to a proxy. Chapter 7 tests this extension through further simulation, to prove that Walkabout can make Internet transfers practical for all mobile devices.

Chapter 6

Extended system design

The basic Walkabout architecture described and tested in the previous chapters partially achieves the goals specified in the introduction of this thesis. It improves transfer times for most message transfers involving mobile devices, and makes it practical for them to transmit larger files. However, it still has some shortcomings which need to be addressed. This chapter covers some extensions to Walkabout that, while not required as part of its fundamental operation, do offer improvements that make it a more powerful and usable system.

The most serious concern is that under certain patterns of consumer movement, the standard model is unsuitable. When a consumer is constantly moving between unique locations, the data overheads are many times the size of the actual message, yet the delivery times are no better than for existing methods such as direct transfers or application-level multicast. This needs to be addressed if Walkabout is to be applied to all mobile situations, so Section 6.1 presents a pre-emptive message delivery service that can make transfers practical under all consumer movement patterns.

Another issue is that the form of addressing specified so far is very low level and only enables transfers directly between devices. Managing lists of other people's devices and their associated keys has the potential to become unwieldy very quickly, so it is far more practical to track a single identity for each person, and let the recipient decide which device should receive it. Section 6.2 proposes a higher level addressing abstraction that allows Walkabout to remain manageable as the number of user devices grow.

One of the main targets of Walkabout are devices which generate content, such as cameras. Even though Table 2.1 shows that these devices do commonly feature some form of wireless networking, they may be closed devices without the option to install additional software. The ultimate goal is that the standard firmware for these devices will eventually

include a Walkabout client, but until such time, it would be desirable to provide some way for these limited devices to connect in to the service. Therefore, Section 6.3 also introduces a standard mechanism to provide system support for limited devices.

6.1 Client manager

As shown through simulation in Chapter 5, a migrating consumer represents the worst-case scenario for Walkabout. The constant movement means that the consumer never returns to a previously visited location, so it never experiences the speed benefits that arise from downloading cached pieces at local speed. However, because the default proxy behaviour is to continue requesting blocks until the consumer reconnects elsewhere, the data overheads can still reach many times the size of the actual message. These problems can be solved by adopting a more proactive approach to proxy downloads. For example, if a migrating consumer's disconnection causes its next visited proxy to start downloading its messages, then it will find pieces waiting when it reconnects, and experience significant improvements in transfer speed. Therefore, this section presents a pre-emptive proxy selection and message delivery service to augment the basic Walkabout model.

A *client manager* is an optional program that operates on behalf of a particular client and coordinates the delivery service. It intercepts any message headers addressed to the client, then follows a *message distribution policy* (MDP) to coordinate the overlay transfers. The MDP determines which proxies should be overlay peers and when they should be downloading the message, in an effort to deliver it to the consumer as quickly as possible. These decisions are made in response to client movement, and may involve a *prediction algorithm* to determine which proxy a client is likely to move to next. Users are able to configure their client managers with whatever combination of MDP and prediction algorithm that they like, and can even choose to modify configurations on demand if their movement behaviour changes. Typical prediction would involve some kind of analysis of a client's movement history, as covered in Section 3.5. Figure 6.1 shows how a client manager's different components fit together in a layered fashion.

A client that has a manager is said to be in *managed* mode. Users are not required to run a manager for their client devices, and if they can not install one or do not wish to have one, clients can easily operate in the regular *unmanaged* mode. Client managers exist to coordinate message delivery to mobile consumers, so if a device sends messages but never receives them (a camera, for example), or if it is stationary, there is no reason for it to have one.

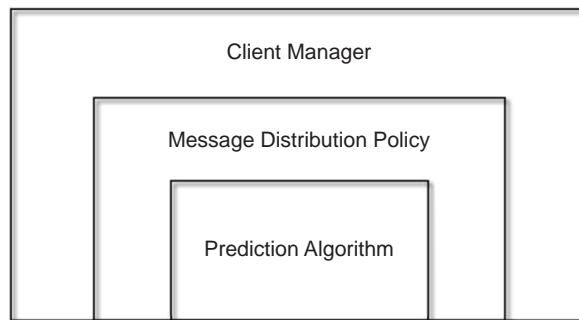


Figure 6.1: The layered client manager components.

6.1.1 Manager creation

A client manager needs to be installed in a stable location, so that it has high availability and is therefore able to build up a solid movement history for its client. It would typically run alongside a proxy on a user's home server, but could also be a stand-alone service running on some other reliable machine.

The manager's role is to assume the identity of the client it is responsible for, so that it can intercept message headers and learn about client movements. The user configures the manager with the public key of the client, causing it to publish the client's key hash to the DOLR at launch. If the manager finds a proxy with an existing publication for its key hash when it launches, it negotiates with them to switch the client from unmanaged to managed mode. It also checks the publication periodically, acting to resume control of the client if a proxy or another client manager has overwritten the publication.

6.1.2 Client connection and disconnection

Any time that a client connects to or disconnects from a proxy, that proxy routes a location update to the client manager. This allows the manager to update the client's movement history, and to start or stop proxy downloads as directed by its MDP.

Because a client manager is optional, a client indicates whether it has one as part of its initial connection to a proxy. If the client does have a manager, the proxy attempts to make contact, and receives headers in response for the messages that it should download. If the proxy is unable to reach the manager for some reason, it publishes the key hash itself and the client reverts to unmanaged mode. However, if the client manager was only temporarily unavailable, it will detect the proxy's publication during one of its periodic checks and negotiate to return the client to managed mode.

When a client in unmanaged mode disconnects, the proxy simply continues downloading their messages without notifying any other host. By contrast, the proxy sends a disconnection notification to the client manager when in managed mode, so that it can coordinate the transfer as necessary and build a record of client movements. This notification includes the last known piece map for each of the client's messages, so that if the manager wishes to initiate a download on another proxy while the client is disconnected, it is able to tell them what pieces the client still needs. The manager may not always receive this notification, possibly due to network issues or the proxy mistakenly thinking that the client is unmanaged. While this can affect the efficiency of the transfer in the short term, normal operation can resume once the manager receives its next notification from the client.

A client may reconnect to the same proxy repeatedly if it is moving between access points on the same network, or if its wireless connection is unreliable. This behaviour can have undesirable effects on the client manager. Firstly, it can pollute the recorded movement history with many connections at the same location, which would then affect the accuracy of future predictions. The other issue is that a momentary disconnection may trigger the download at another proxy, which would adversely affect transfer performance if the client was not actually moving.

The simplest solution is to set a time threshold at the proxy, to prevent it from reporting a disconnection unless it is over a minimum length (which would be in the order of several seconds). When tuned to the specific network conditions, this would be expected to detect most instances of momentary disconnection. This approach requires caution, because performance could be affected if notifications for real movements are delayed due to an unnecessarily long threshold. Alternatively, the proxy could use knowledge of the physical access point location and recent client movement history to decide when to notify the client manager. For example, if the client disconnects from an access point on the top floor of its building, the proxy might wait 30 seconds before notifying the client manager; but if the client disconnects from the foyer, and its recent movement has been in the direction of the exit, then the proxy may notify the client manager instantly.

6.1.3 Prediction algorithm

The prediction algorithm analyses the information available at the client manager to determine where the client is likely to connect in the future. This evidence is typically drawn from the proxy connection history of the client, which the client manager collects automatically as the client moves. It could also be supplemented by other sources, such as diary appointments

or an external tracking service that provides the user's real-world position. The ideal algorithm would analyse this evidence to decide upon a single proxy with 100% certainty every time, but this is unlikely to be attainable in practice.

Markov predictors, as discussed in Section 3.5, are a simple yet effective way to predict movement patterns based upon historical evidence. Therefore, a Markov-based prediction algorithm is used to assess the viability of prediction-based client managers in Chapter 7.

6.1.4 Message distribution policy

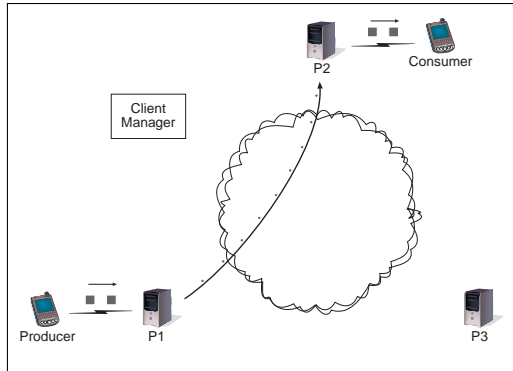
The message distribution policy (MDP) dictates the message download behaviour of proxies, so that the client manager's consumer has the largest possible pool of pieces to download from at high speed while it is connected. The faster the consumer receives pieces, the less time it takes to complete the entire message transfer. The MDP is called upon to make these decisions in response to client connection and disconnection notifications, and upon the receipt of new message headers. It would usually, but not always, be informed by a prediction algorithm.

Any time the MDP triggers a particular message download at a proxy, the manager sends across the header, with instructions to start the download immediately. The header is also accompanied by the client's last known piece map, so that the proxy knows what the client pieces still needs. The proxy uses this information to join the overlay, if it is not already a member, and to start requesting blocks from the overlay peers.

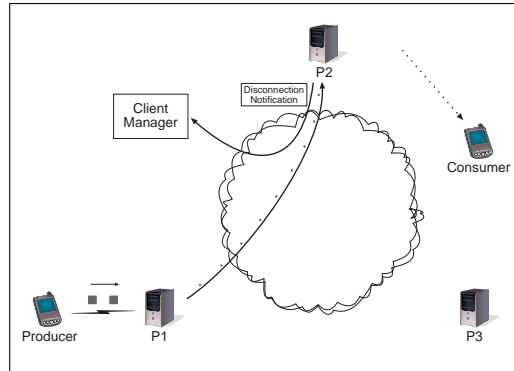
The client manager also has the ability to tell proxies to stop downloading a message on behalf of a consumer. One reason it might do this is if the client just disconnected from that proxy. The other reason would be if an incorrect prediction caused the client manager to start the download at one proxy, but the client actually connected to a different one. The proxy is not required to stop the download, as it may wish to continue downloading on behalf of other consumers. If the proxy stops downloading it still remains as an overlay member, so that can improve future transfer times by offering any pieces it has to its peers.

The most promising MDP is *predict-on-disconnection*, as seen in Figure 6.2, because it has the potential to provide moving consumers with the local caches that are usually only available to reconnecting consumers. Under this policy, the client's current proxy is the only one that downloads the message while the client is connected. When the manager learns that the client has disconnected, it consults the prediction algorithm to determine where the client is most likely to move to, and prompts that proxy to start downloading the message. It also tells the proxy that is currently downloading to stop. If the manager receives any new message

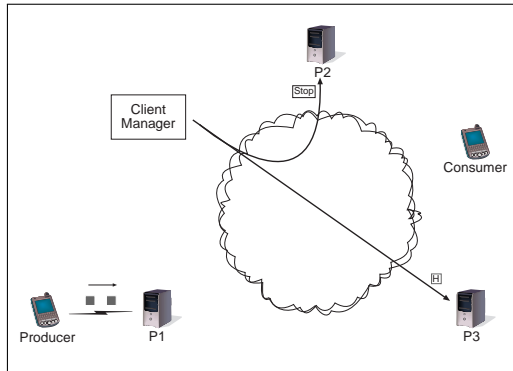
6.1. Client manager



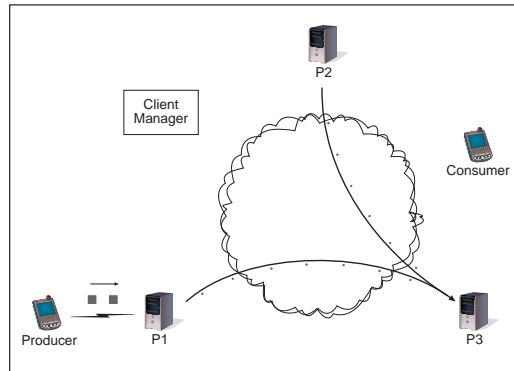
(a) The consumer is initially connected to P₂, which is downloading blocks from P₁.



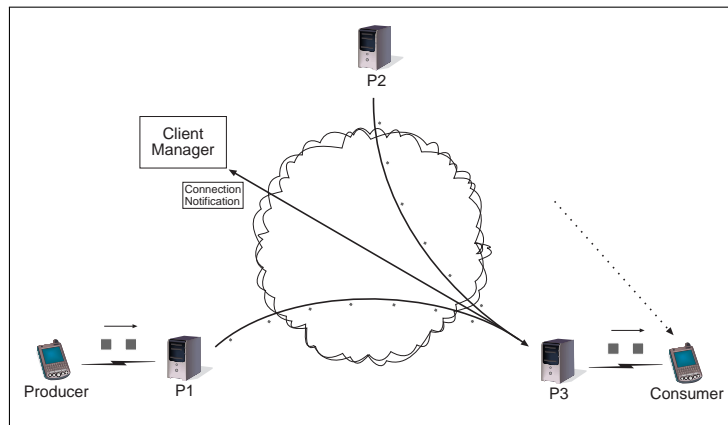
(b) The consumer disconnects, and P₂ informs the client manager.



(c) The client manager predicts that the consumer will move to P₃, so it sends the header to P₃ and tells P₂ to stop downloading.



(d) P₃ starts downloading blocks from P₁ and P₂.



(e) The consumer connects to P₃ and starts downloading cached pieces immediately. P₃ tells the client manager that the consumer has connected.

Figure 6.2: The predict-on-disconnection message distribution policy in action.

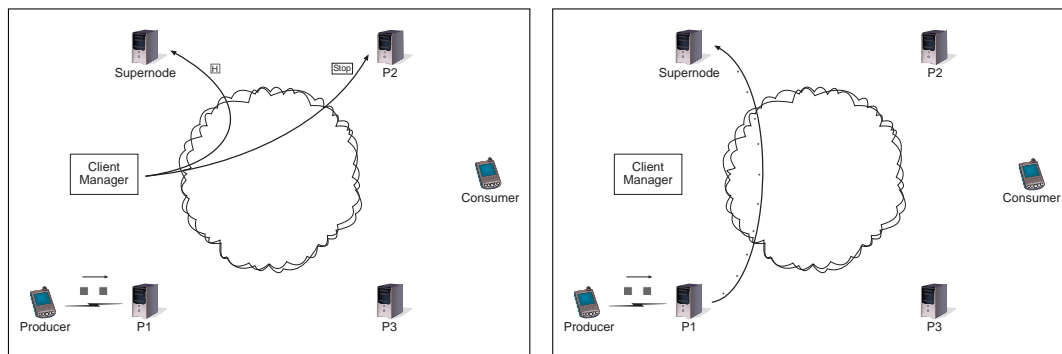
headers while the client is in transit, it forwards them to the predicted proxy too. The hope is that if the prediction is correct, the client will connect to the proxy to find that there are already pieces waiting in the local cache, that can be downloaded at local speed. If the client connects to a different proxy instead, the client manager stops the download on the incorrectly chosen proxy, and starts it on the correct one.

A *predict-on-connection* MDP might be expected to further improve transfer speeds, at the risk of greater data overheads if prediction is incorrect. When the client connects to a proxy, the manager immediately starts the download at the next predicted proxy, so that both proxies are downloading simultaneously. The intention is that by the time the client connects to that predicted proxy, there should be a bigger buffer of message data waiting than if the download had only started when the client disconnected. In practice, though, this is unlikely to happen. If the downlink speeds are greater than the uplink speeds, as is typical, any pieces that the predicted proxy completes will be immediately advertised to and retrieved by the consumer's current proxy. The client will download all the unique pieces that the predicted proxy acquires during this time, and so there will never be any benefit gained by splitting the source proxy's limited uplink speed between multiple peers. Better performance would be achieved by simply concentrating on delivery to the client's current proxy, as in the *predict-on-disconnection* policy. This method would be beneficial, though, if the source proxy has an uplink speed that is greater than the destination downlink speeds. This allows it to simultaneously saturate the downlink of both the current and predicted proxies, so that the predicted proxy is able to build the buffer of unique pieces that the MDP aims to create.

These prediction-based policies can be extended so that any number of proxies are selected each time the client connects or reconnects. This could be useful if the manager is unsure where the client is heading, or wishes to start the downloads even further in advance to gain a greater advantage. This has the potential to improve transfer times, but only under favourable network conditions, and at the cost of great overheads. A more sophisticated approach is to offer different client piece maps to the proxies, where the proxies with the highest probabilities are asked to download more, to limit the amount of unnecessary data downloads.

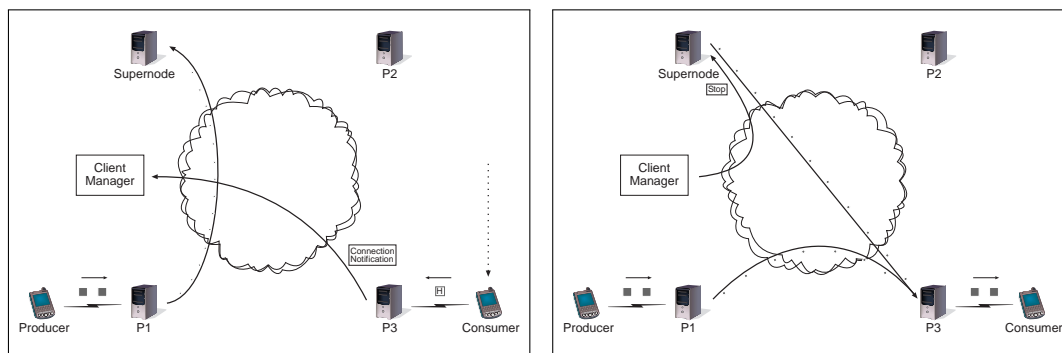
The *supernode* MDP, as illustrated by Figure 6.3, is an alternative on-disconnection policy that does not involve any prediction. Instead, the client manager introduces a proxy with a very fast uplink and downlink to the overlay, that will never contact a client directly. This is the supernode. When the client is connected, its current proxy is the only one that downloads message data, but when it disconnects, the supernode becomes the only proxy that does. This causes the supernode to build up a cache of pieces, so that when the client reconnects, its new proxy can obtain blocks rapidly from both the source proxy and the supernode.

6.1. Client manager



(a) In response to client disconnection, the client manager send the header to the supernode and tells P_2 to stop downloading.

(b) The supernode starts downloading blocks from P_1 .



(c) The consumer connects to P_3 and uploads the header. P_3 tells the client manager that the consumer connected.

(d) The header causes P_3 to start downloading blocks from P_1 and the supernode. The client manager tells the supernode to stop downloading.

Figure 6.3: The supernode message distribution policy in action.

This MDP is most effective when proxy downlink speeds are faster than peer uplink speeds, and when the consumer is moving rapidly in an unpredictable pattern. Even though the consumer will never encounter a local cache of pieces when it connects to a proxy, it is still able to obtain pieces at a faster rate than if it were unmanaged. This is because access to the supernode means that its proxy can download blocks on demand at a rate limited by its own Internet downlink, rather than the source proxy's uplink. The data overheads are also kept under control, because the only proxies that ever download the message are those that the consumer is currently connected to, or the supernode. A drawback of this approach is that it requires access to a fast server, which may be difficult to obtain, unless it is provided by an ISP, a large organisation such as a university, or a web company like Google (2007c) or Yahoo! (2007c).

There is great scope to explore the impact of different MDPs on the speed and efficiency of managed Walkabout transfers. The predict-on-disconnect and supernode policies

should provide the most benefits under general conditions, and so the next chapter evaluates them through simulation.

6.1.5 Message completion

Once a consumer completes or rejects a download, the proxy informs the client manager, so that it knows to delete all state associated with the message. It maintains the record of client movements that occurred during the delivery, as these can help to make more accurate predictions in the future.

6.1.6 Additional functionality

There is the potential that, just like email today, unsolicited Walkabout messages could become a problem. Even if the consumer constantly rejects them, they could still lead to significant numbers of message headers being delivered to consumers, wasting the time that could be used to download messages that they want. In addition, if a proxy receives a header for a client that is disconnected, it will assume that they will want the entire message until it learns otherwise, so these unsolicited message could result in significant amounts of wasted Internet bandwidth at the proxies, too.

Because the client manager is the first point of contact for all messages addressed to a managed client, it is in the perfect position to fight unsolicited messages. In a similar fashion to email filtering, it could examine the metadata within the message header to determine whether it believes its client will be interested, and forward or drop the header accordingly. Consumers could also create their own filtering rules on the client manager.

It should be noted that short messages are also intercepted by the client manager. This type of message is intended for real-time communications, so the client manager only forwards them if the client is connected at the time. Otherwise, it simply discards them.

6.2 Identity tracker

One problem with the basic Walkabout message delivery model is that when a producer wishes to send a message, it needs to specify the exact devices that are the consumers. This is manageable when a user configures communications between their own devices, but can become impractical when one user wishes to send a message to another. The person sending the message may not always know the best device to contact the recipient on at that point in

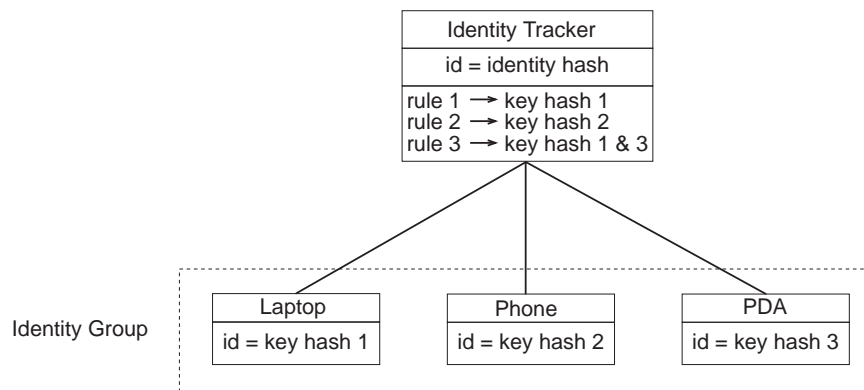


Figure 6.4: An identity tracker and its identity group.

time, or the specific key hash required. This approach is also inflexible, as the recipient has no way of indicating which devices they would actually prefer to receive messages on.

An example of a more desirable addressing model can be seen with email. The sender simply specifies the recipient's email address when they send the message. When the recipient chooses to check their email, they access it using the most practical device at the time, which could be a mail application on their desktop or laptop computer, a web mail interface on a friend's computer, their phone or their PDA. This is both simple for the sender, and flexible for the recipient.

This section presents the *identity tracker*, an extension to Walkabout that provides a high-level addressing abstraction. It enables a user to unify their devices under one globally addressable key hash, so that the sender only needs to know one identity to send them a message. The user specifies which devices they would like to receive which messages, and can choose to multicast them for delivery to multiple devices. Figure 6.4 outlines the relationship between a sample identity tracker and the devices that it represents.

6.2.1 Tracker creation

A user that wishes to use an identity tracker generates a key pair to represent their global identity. They load the pair on to any device that is part of the *identity group*, as well as on the identity tracker service, which is installed in a stable location. The identity tracker publishes the hash of the identity's public key (the *identity hash*) to the DOLR. Finally, the user distributes their public key to anyone who may wish to contact them.

6.2.2 Device and rule registration

The user establishes their identity group by registering the keys of their individual devices with the identity tracker. One of the advantages of this approach is that it centralises all of the device public keys, so that nobody else needs to know what they are. The user can add or remove devices, or even modify the keys, and messages will still be delivered appropriately without the sender having to update anything.

A set of rules determines how different messages are distributed to devices. Each rule specifies a simple expression that maps message header metadata to a list of identities. Any of the standard header fields can be used in the expression, and applications can specify their own tags as part of the freeform application data field. These rules are similar in concept to the subscriptions used by content-based publish/subscribe messaging systems, so a filter language based on the one that Elvin uses (Mantara Software, 2003) would be suitable, for example.

The identity tracker applies the rules to any message headers that it receives, and forwards them to the desired devices. For example, a user could specify that they wish to receive all messages from their friends that are less than 1MB on their mobile phone, but that all others should be directed to their laptop. The language that describes these rules could also be extended to incorporate time spans, so that certain rules only apply at certain times. As another way of combating unsolicited or unsuitable messages, a rule may instead specify to drop a message.

The user can add, query, modify or delete group membership and rule sets from any device that is part of the identity group. Every device has a copy of the identity's private key, so it is used to sign the update notification. The tracker checks the signature before processing the update, and only proceeds if it is able to verify it.

6.2.3 Delivering messages

To send a message to an identity group, a producer simply specifies the identity hash as one of the consumers and uploads the message to a proxy. It can also use the identity's public key to encrypt the message. The local proxy then routes the header to the identity tracker. From the point-of-view of the producer and the proxy, sending a message via an identity tracker is identical to sending one directly to a device. However, an identity tracker is provided primarily as a convenience, so producers are not required to route their messages through them. All of the devices retain their original key hashes, allowing a producer to instead route to them directly if it knows the appropriate key.

When the identity tracker receives the header, it applies the rule set to the metadata within the message header to determine the real consumers for the message. Depending on how many matches there are, the output set can contain key hashes for zero, one, or multiple consumer devices. If there are any matches, it associates the key hashes with the identity hash in a new *identity mapping* header field (which is excluded when calculating the message signature, because it can not be determined at the time of message creation). If there are no matches, or the message is explicitly ignored, the identity tracker routes an immediate rejection notification to the message tracker.

Once the modified header reaches a proxy, the delivery process is the same as in the standard model, with the exception that the proxy looks in the identity mapping as well as the standard list of consumers to determine which local clients it is delivering the message to. When a client completes or rejects a message, the proxy returns the appropriate notification to the identity tracker denoted by the identity mapping. The identity tracker aggregates all of the responses, and routes a single notification to the message tracker when all consumers have replied.

6.2.4 Groups

Identities can be aggregated into groups, to allow entire families or organisations to be represented by a single identity. Each group member has a copy of the group key pair, which they install on their devices, to enable group modification and message decryption. Groups are managed by a regular identity tracker, which maintains registrations of the identity hash for each of the users. A message header sent to the group is initially routed to the group's identity tracker, but is forwarded on to other identity trackers according to the rule set, until it ultimately reaches the consumers. If a proxy receives multiple copies of the same header as a result of these branching forwarding paths, it merges the identity mapping fields.

Groups can be nested within other groups to indefinite depth, but there are limitations to this approach. Each DOLR delivery takes time in the order of seconds, so a long propagation sequence could add up to a substantial amount of latency between when the producer sends the header, and when it reaches each of the consumers. Also, while Walkabout has been shown to scale well to a small number of consumers, it is not designed for a large number. Multiple groups could lead to a very large number of final consumers, at which point some other delivery mechanism would be a better choice. Finally, care also needs to be taken to avoid cycles between identities.

Despite its limitations, the use of groups has the potential to offer a useful facility to Walkabout, if it is managed carefully.

6.2.5 User agents

While the basic approach is for the user to manually select how to direct messages to devices, a more practical approach could be for an intelligent agent to do so on their behalf. By looking at the user's location (possibly through communication with the device client managers), an agent could modify the rules without direct input from the user, so that they receive their messages in the most appropriate manner. This agent could be an external program which issues rule changes to the identity tracker, or part of the identity tracker itself, such that it forwards messages based upon more a complex decision process than a simple set of user defined rules.

6.3 Simple device agents

Walkabout is an effective form of communication for wireless mobile devices, but it may not always be possible to install the client software on the device. Therefore, there are some types of devices that would benefit greatly from the service that are unable to access it, such as Bluetooth or WiFi enabled cameras and media players (refer to Table 2.1 for the networking capabilities of some common devices). This section presents the design for a *device agent*, a optional system component that provides a bridge between limited devices and Walkabout. It enables these devices to register their location, send messages to a location for backup, and receive messages from any client. The agent may run on the same machine as the local proxy, or it may be a separate node with its own wireless networking interface.

Different WiFi devices use different communication protocols. Sometimes, they will only communicate with a particular piece of software, as with the Kodak EasyShare-One camera range or the Microsoft Zune. This makes it difficult to generalise agent functionality, so the design of the device agent relies upon plugins to provide support for different devices. However, Bluetooth specifies standard protocols to exchange objects via the object push and file transfer profiles, and these are widely supported. This section concentrates on how a device agent communicates with Bluetooth devices; any future plugins for WiFi devices should attempt to follow the same approach.

When a Bluetooth-enabled device wants to use Walkabout, it simply searches for and pairs with an available device agent in its environment. The device can not be expected to have a public key, so the MAC address of its Bluetooth interface forms the basis of its identity instead. The agent generates a registration notification that uses the hash of this MAC address as the identity, then sends it to the local proxy in an manner that is analogous to a regular

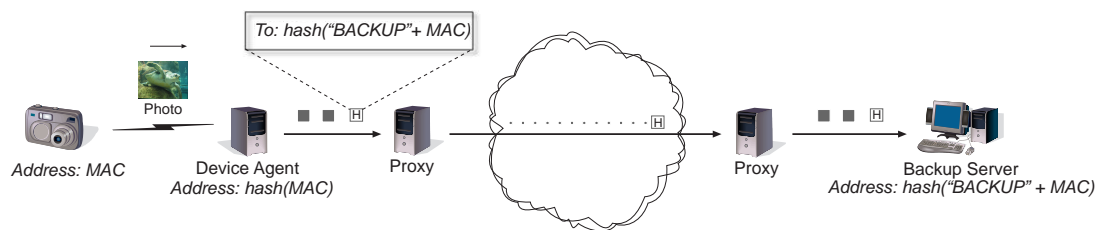


Figure 6.5: A simple device uploading to its backup server.

client-proxy registration. The proxy recognises the agent as a client, and publishes the hash to the DOLR. The agent also notifies the proxy if it loses contact with the device.

To upload a file via Walkabout, the Bluetooth device simply pushes it to the device agent using the regular Bluetooth object push or file transfer protocol. Unfortunately, there is no reliable way for the device to specify Walkabout metadata with the file, so it is restricted to a single destination – its dedicated backup server. This is a specialised Walkabout client that uses a *backup hash* as its identity, derived by concatenating the string “BACKUP” and the client’s MAC address, then hashing the result. When the agent receives a file from the device, it generates the header and specifies the calculated backup hash as the message consumer. It then breaks the file in to pieces and uploads them to the proxy. The subsequent transfer to the backup server follows the regular Walkabout protocols, although the proxy does not forward the final delivery notifications to the uploading device. Figure 6.5 summarises the upload process.

By contrast to the upload, a limited device is able to receive messages from anybody (provided that the device actually supports file downloads). The device agent is registered under the device’s identity, so it receives Walkabout message headers and pieces on their behalf, as if it were a regular Walkabout client. When the message is complete, it simply pushes the file to the device if it is still available. There is no way for a simple device to recognise message pieces, so if a transfer to or from a device is interrupted, it must restart from the beginning upon reconnection.

This approach does not take full advantage of Walkabout’s features, but it does allow simple devices to make use of the location independent addressing, so that it is easy for them to backup their files and receive downloads across the Internet.

6.4 Summary

The extensions covered in this chapter provide a number of ways to improve the performance, flexibility and availability of Walkabout. In particular, the client manager provides a pre-emptive message delivery service that has the potential to overcome the performance problems with the core architecture that Chapter 5 identifies. Therefore, the design for the client manager forms the first part of Contribution C3 from Section 1.4. The next chapter simulates Walkabout deliveries when client managers are available, to measure how different MDPs and prediction algorithms affect performance.

Chapter 7

Client manager evaluation

Chapter 5 shows that Walkabout is a viable message transfer architecture in many instances, but that it generates slow delivery times and high data overheads when consumers are constantly moving between networks. This chapter uses simulation to provide evidence that the client manager system extension proposed by Chapter 6 addresses these shortcomings. It analyses transfer performance when client managers are configured with different message distribution policies (MDPs) and prediction algorithms, and then compares them to each other to determine which approaches perform best under which conditions.

7.1 Experimental setup

The experiments in this chapter were designed to test the performance of Walkabout transfers directed by client managers, and they were constructed once again using the OMNeT++/INET framework. They explored client managers that use predict-on-disconnection and supernode MDPs, and evaluated them using the primary metrics of consumer download speed and data overheads.

7.1.1 Test network

These experiments were built upon the same simplified Internet configuration that was used in Chapter 5. Asymmetric network links are common in home Internet connections, so all sub-networks were provided with a 500KB/s downlink and 100KB/s uplink.

A client manager's role is to improve performance for mobile consumers, so the test network featured fixed producers and mobile consumers. Each consumer was randomly allocated a home proxy, which was responsible for creating a client manager and publishing it

to the DOLR service during network initialisation. Consumer connection periods were always fixed at 1 minute, while disconnection periods varied between 10 seconds and 30 minutes. Depending on the experiment, a consumer followed one of three different movement patterns:

- *reconnecting*, where it connects back to the same proxy after every disconnection;
- *migrating*, where it only reconnects to a proxy it has not visited previously, selected using a uniform random distribution;
- a *real-world* pattern based upon the traces from Dartmouth College.

All message transfers took place between a single producer and consumer, with message sizes ranging between 50MB and 200MB. The lower bound was raised from the 5MB that was used in previous simulations, to increase the average delivery times and thus the number of proxies that a consumer visits during a transfer. This was to provide a clearer picture of the effect that prediction has, where appropriate. Table 7.1 summarises the experimental parameters used in this chapter.

7.1.2 Evaluation criteria

The main role of the client manager is to coordinate proxy downloads so that any time a consumer connects, there are pieces waiting for it to download at high speed. Therefore, the *average consumer download speed* is the best indicator of how effective an MDP is. It is often desirable to attain these speeds without generating too much excess data, so the metric of *average data overhead* (as defined in Section 5.1.2) is also important.

In Section 5.3.4, the effective transfer speeds when delivering a message to an unmanaged reconnecting consumer were found to be close to those experienced by a fixed consumer, and were therefore considered practical. This performance was due to the high download speeds that the consumer experienced upon reconnection. In this chapter, those consumer download speeds are used as a benchmark, so if a client manager enables a consumer with a different movement pattern to achieve similar results, its performance will also be considered practical.

The predict-on-disconnection MDP requires a prediction algorithm. Any time that a client disconnects, the MDP queries this algorithm to predict which proxy it will connect to next. The *prediction accuracy* describes how suitable the algorithm is to a given set of

Table 7.1: Experimental parameters for Chapter 7

Exp.	Total Client Count	Producer Mobility Pattern	Consumer Mobility Pattern	Downlink Speed (KB/s)	Uplink Speed (KB/s)	Conn. Time (s)	Disconn. Time (s)	Message Size (MB)	MDP	Prediction Type	Results
1	50	Fixed	Migrating, reconnecting	500	100	60	10–1800 (varying)	50–200 (random)	Predict-on-disc., none	Perfect	Figure 7.1, Figure 7.2
2	50	Fixed	Migrating, real-world	500	100	10	1800	200	Predict-on-disc., none	Perfect	Table 7.2
3	50	Fixed	Real-world	500	100	60	10–1800 (varying)	50–200 (random)	Predict-on-disc.	Fixed 0–100% (varying)	Figure 7.3, Figure 7.4, Figure 7.5
4	50	Fixed	Real-world	500	100	60	10–1800 (varying)	50–200 (random)	Predict-on-disc., none	Perfect, Markov, fixed 57%, fixed 83%	Figure 7.8, Figure 7.9, Figure 7.10
5	50	Fixed	Migrating, real-world	500 (proxy), 1GB/s (s/node)	100 (proxy), 1GB/s (s/node)	60	10–1800 (varying)	50–200 (random)	Predict-on-disc., supernode, none	Markov	Figure 7.12, Figure 7.13
6	50	Fixed	Migrating, real-world	1024 (proxy), 1GB/s (s/node)	100 (proxy), 1GB/s (s/node)	60	10–1800 (varying)	50–200 (random)	Predict-on-disc., supernode, none	Markov	Figure 7.14

conditions, and it is measured across all clients for the duration of a message by:

$$\text{Prediction accuracy} = \frac{\text{Total number of correct predictions}}{\text{Total number of predictions made}}$$

A prediction is only made if the client manager has sufficient evidence to guess where the client is headed; otherwise no prediction is offered, and the accuracy remains unaffected when the consumer next connects.

As with Chapter 5, the results in this chapter are only indications of performance. They show the trends as conditions change, but do not always translate to general conditions. This is particularly evident when considering the exact points at which one approach becomes more attractive than the alternatives.

7.2 Predict-on-disconnection

The predict-on-disconnection MDP is used by the client manager every time a consumer disconnects. It aims to improve transfer performance by predicting where the consumer will move to, then starting the message download at the appropriate proxy as soon as possible. The intention is to have numerous pieces waiting in the proxy's cache by the time the consumer connects, which it can then download at local speed. The success of this MDP depends upon the accuracy of its prediction algorithm. This section begins by looking at the results under two theoretical prediction algorithms: one that always makes the correct decision, and one that makes the correct decision with a varying degree of accuracy. These are then compared to a Markov-based algorithm, which uses historical evidence to predict future movement.

7.2.1 Perfect prediction

As has been demonstrated previously, an unmanaged Walkabout transfer performs best when a consumer is reconnecting, because every piece downloaded in the consumer's absence is available for it to download at local speed immediately upon reconnection. A client manager configured with the predict-on-disconnection MDP seeks to reproduce this behaviour under all movement patterns, so that the consumer will always find the same amount of data waiting for it, regardless of where it moves to. Figures 7.1 and 7.2 compare the performance of an unmanaged reconnecting consumer (which exhibited the best performance in Chapter 5), an unmanaged migrating consumer (which exhibited the worst performance) and a migrating consumer that is managed by a perfect predict-on-disconnection MDP. Figure 7.1 shows that

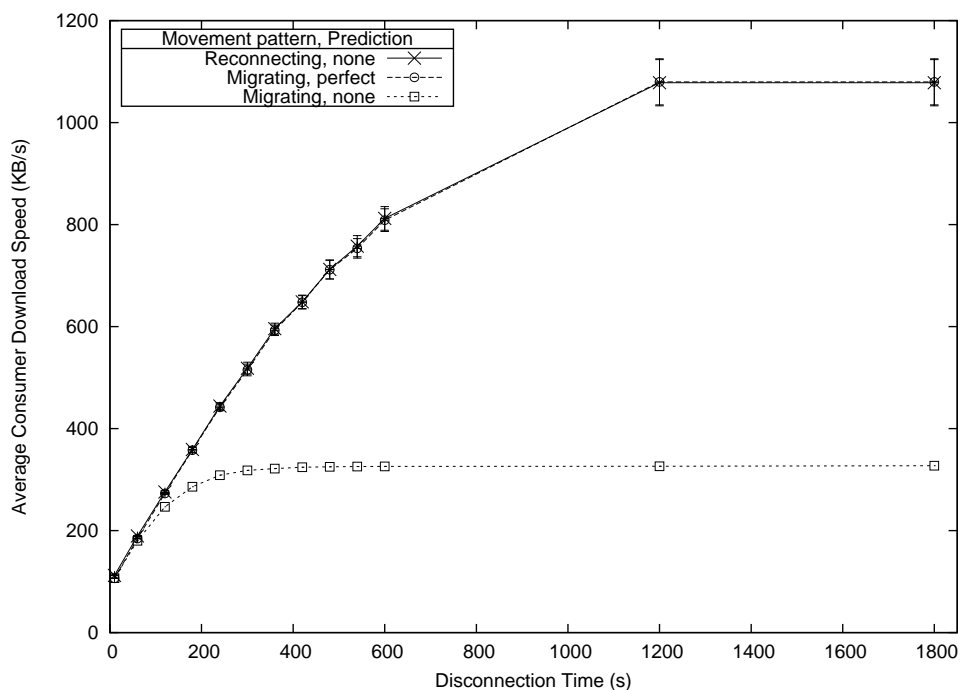


Figure 7.1: Impact of perfect prediction on the average consumer download speed.

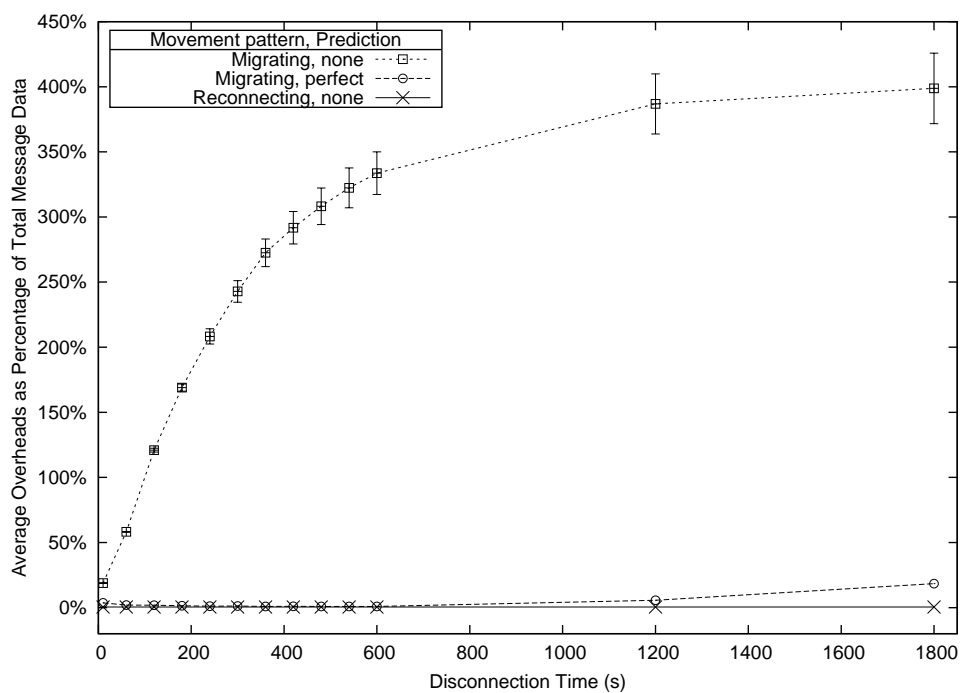


Figure 7.2: Impact of perfect prediction on average data overheads.

7.2. Predict-on-disconnection

Table 7.2: Experimental results for a 200MB transfer using perfect prediction, with 10 second connection and 30 minute disconnection periods.

Movement pattern	Avg. consumer download speed (KB/s)	Avg. overheads
Migrating	1723.2	510.2%
Real-world	1738.5	137.0%

when this MDP is used, a client manager improves the migrating consumer's average download speed so much that it becomes indistinguishable from that of an unmanaged reconnecting consumer.

The same is true of the message overheads presented in Figure 7.2. Under most disconnection periods, overheads are close to zero for both the managed migrating and unmanaged reconnecting consumers, which is a significant improvement upon the levels shown for an unmanaged migrating consumer. There is a slight increase in overheads for the managed consumer when the disconnection periods become so long, compared to the connection periods, that each proxy downloads more data than it can actually deliver to the consumer while it is connected, even with the 2MB/s bandwidth that is available. There is a high chance that the consumer will instead acquire the remaining pieces from a different proxy, so they will never be delivered locally and thus become transfer overhead. The overheads seen here are only of minor concern, as they remain quite low and are still significantly less than for an unmanaged migrating consumer. If disconnection periods were to grow very long, and connection periods were correspondingly very short, then overheads could climb quite high, with each proxy downloading nearly the entire message unnecessarily. The migrating consumer is an extreme case, so Table 7.2 compares the performance to what a real-world consumer experiences, for a 200MB transfer with 10 second connection and 30 minute disconnection periods. It shows that while these conditions can lead to very high overheads in both instances, they are reduced when movement patterns are more realistic, because some proxies have multiple opportunities to transfer their pieces.

Fortunately, this extreme relation between connection and disconnection time is rare enough that it can be safely ignored in the experiments presented here. If a consumer was following this sort of pattern regularly, then the MDP could be modified to limit the number of pieces that the predicted proxy caches. This could be achieved by delivering piece maps that only contain a subset of the consumer's needs, or by predicting connection time as well as movement sequence and restricting the amount of time the proxy is asked to download for. These approaches would restrict the amount of data the proxy downloads in a consumer's absence, but at the risk of missing out the potential performance gains if the consumer ever

connects for a longer period. If overheads are excessive, this trade-off may be considered worthwhile.

These results prove that a client manager running a predict-on-disconnection policy has the potential to provide high download speeds and low overheads to a Walkabout consumer. Under most conditions, perfect prediction provides the same performance as the best-case unmanaged reconnecting consumer. The overheads increase if consumer disconnection periods are long and connection periods are short, but only enough to be of concern in extreme cases.

Unfortunately, perfect prediction is difficult to obtain, so this level of performance would be unlikely in practice. These results do provide an upper level to strive for, though, and a decent prediction algorithm should be able to get close. Later in this chapter, an approach based on Markov chaining shows this to be true.

7.2.2 Imperfect prediction

In reality, a prediction algorithm would be expected to make mistakes. If it is good, then these mistakes will not occur very often and the prediction accuracy will remain high. When accuracy falls, transfer performance will probably fall too. This experiment examines the relationship between prediction accuracy and transfer performance, by configuring the client managers to use a variable accuracy prediction algorithm. The consumers follow real-world movement patterns, and whenever one disconnects, the algorithm either correctly returns the next proxy it will connect to, or an incorrect one chosen randomly from those that it has visited previously.

Figures 7.3 and 7.4 outline the results of simulating transfers with varying levels of prediction accuracy. Figure 7.3 shows that consumer download speed decreases gradually with accuracy, due to the diminishing chance that a consumer will find cached pieces waiting when it connects to a proxy. The effect is less pronounced when disconnection time is low, because even a correct prediction will only provide proxies with limited opportunity to build piece caches. Figure 7.4 shows that there is also a correlation between decreasing accuracy and increasing overheads, as a result of the increased chance that a proxy will be asked to pre-emptively download pieces on behalf of a consumer that does not arrive.

The most interesting points on these two plots are where their performance intersects with the observed values for an unmanaged real-world consumer. This gives the minimum prediction accuracy under these network conditions where it becomes worthwhile to have a client manager running the predict-on-disconnection MDP. Figure 7.5 compares the results for an unmanaged consumer to the closest slices taken from 3D plots of Figure 7.3 and Figure 7.4. It

7.2. Predict-on-disconnection

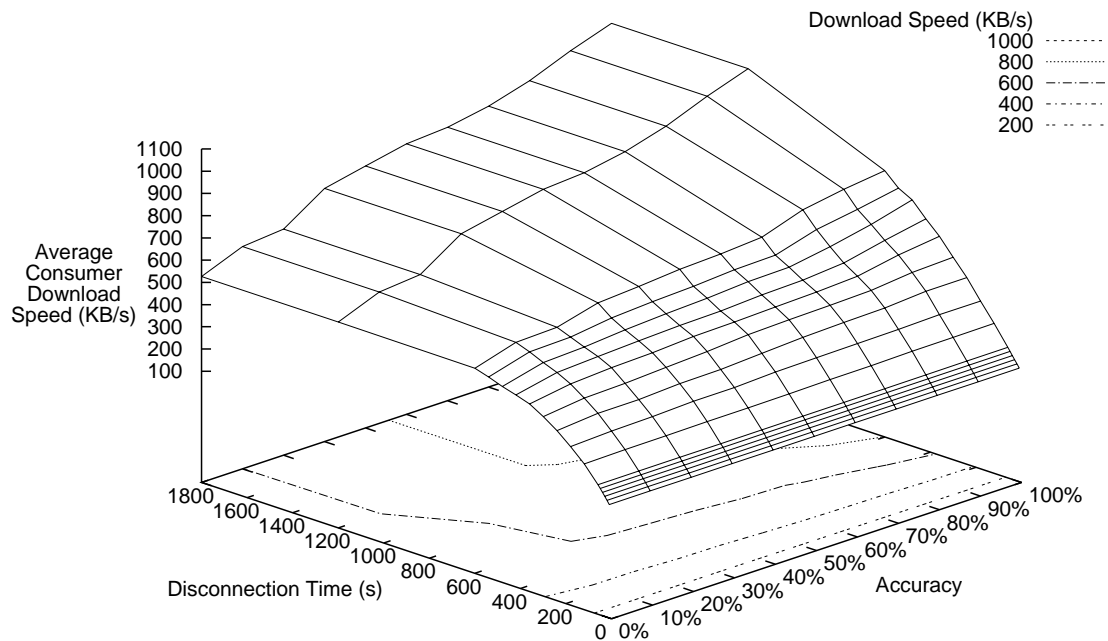


Figure 7.3: Impact of varying prediction accuracy on the average consumer download speed. Line intersections correspond to recorded data points.

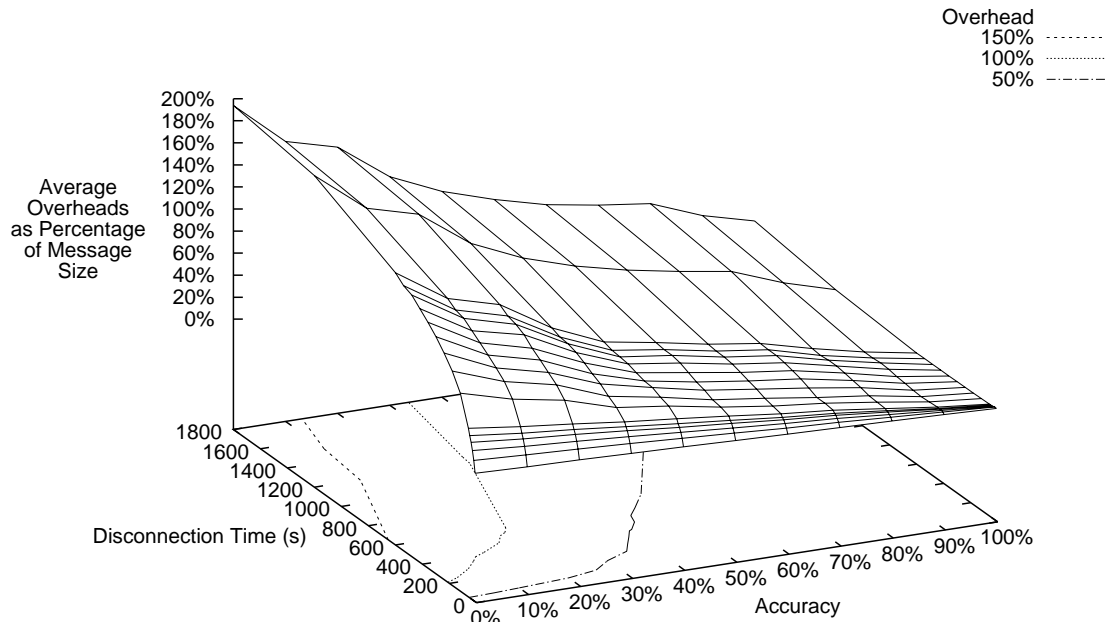


Figure 7.4: Impact of varying prediction accuracy on average data overheads.

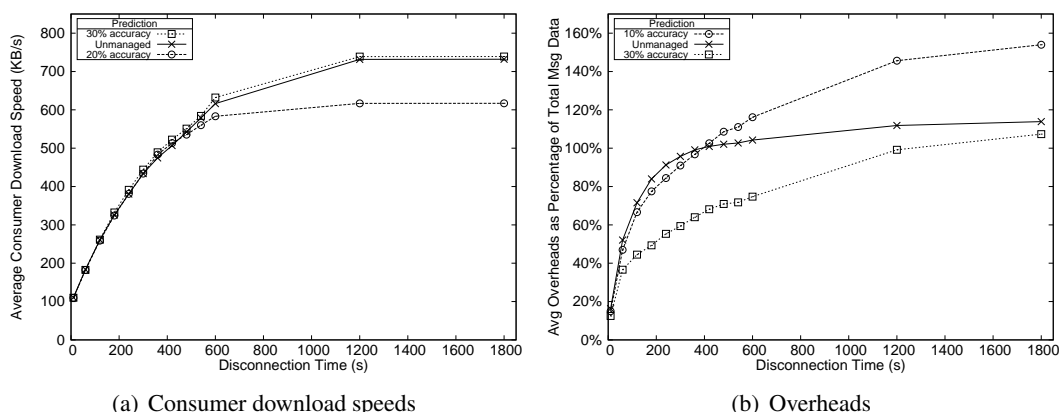


Figure 7.5: Crossover points between imperfect prediction and unmanaged transfers.

shows that consumer download speeds for an unmanaged consumer correspond to those experienced by a managed consumer with fixed accuracy in the range of 20-30%, and Figure 7.5(b) shows the similar range of 10-30% for overheads. These crossover points are not exact, but do reveal two important properties. The first is that a very low accuracy prediction algorithm can harm transfer performance, so if a consumer’s movement is very unpredictable, an MDP other than predict-on-disconnection should be used. The other property is that the crossover point is quite low, at around 30%, which indicates that even a mediocre prediction algorithm may be sufficient to extract some performance boost over an unmanaged transfer.

7.2.3 Markov prediction

While the predict-on-disconnection results presented so far in this chapter give an indication of performance, the prediction algorithms used were purely theoretical. This section presents a concrete example of a prediction algorithm that could be deployed in Walkabout client managers, and an analysis of how effective it is.

The proposed algorithm uses $O(2)$ Markov prediction with fallback (as covered in Section 3.5) to select proxies based upon a client’s historical movement patterns. If multiple proxies are found to be equally likely, then the most recently visited one is selected. This approach has previously been used to predict device movement on an earlier version of the Dartmouth traces, with moderate success (Song *et al.*, 2006). It yielded the best results in comparison to other $O(k)$ Markov and Lempel-Ziv-based (Ziv & Lempel, 1978) methods, with a median prediction accuracy of 72% when only those traces with more than 1000 transitions between connection points were considered. The results for imperfect prediction from the previous section suggest that this level of accuracy should be enough to significantly improve consumer download performance beyond the unmanaged model.

As Chapter 5 discusses, the Dartmouth traces were simplified for the purposes of these simulations, such that they only represent movements between buildings. Therefore, the Markov algorithm was applied to the modified traces to determine how accurate it is at predicting device movement patterns. Only those traces with more than 100 transitions between buildings were considered, because the simplification reduced the size of the traces. The resultant distribution of prediction accuracy can be seen in Figure 7.6. For the 4293 eligible devices, the median prediction accuracy was reduced to 55%, while the average was 57%. This is quite low, but it remains higher than the minimum accuracy of 30% required to justify a client manager. The process was then repeated for the 50 device traces containing the highest number of transitions, as these most populous traces were the ones used in this chapter's real-world experiments. As Section 5.3.4 explains, these particular traces were chosen to thoroughly test Walkabout's performance. The results presented in Figure 7.7 reveal a much higher degree of accuracy when predicting their movement, where the median was 87% and the average was 83%.

The degree of prediction accuracy obtained from the preliminary experiments was promising, so the next step was to incorporate the algorithm in a client manager. The following experiment simulated a series of message transfers between clients following the real-world movement patterns. Each client manager was equipped with the predict-on-disconnection MDP and the Markov prediction algorithm. The prediction algorithm processed the first 1000 movements of its client trace during initialisation, to build up an adequate movement history. Each client then began its movements at the first unprocessed entry. These transfers also were repeated with imperfect prediction algorithms, fixed at 57% and 83% accuracy, and for unmanaged consumers.

Figure 7.8 presents the consumer download speeds for this experiment, and Figure 7.9 presents the data overheads. The Markov algorithm shows that it is a reasonable prediction option, as it produces consumer download speeds that approach those of perfect prediction. Average data overheads also remain below 40%, even during the longer disconnection periods that cause an unmanaged transfer to have overheads in excess of 120%. These high overheads were one of the major problems with the basic Walkabout model, so this improvement is significant.

The 83% fixed accuracy predictor approximates the Markov algorithm for the top 50 clients, as shown by the similarity between the two sets of results. This implies that the 57% fixed accuracy predictor can be considered a fair approximation of how the Markov algorithm would perform if it were applied to the larger set of available client traces. This

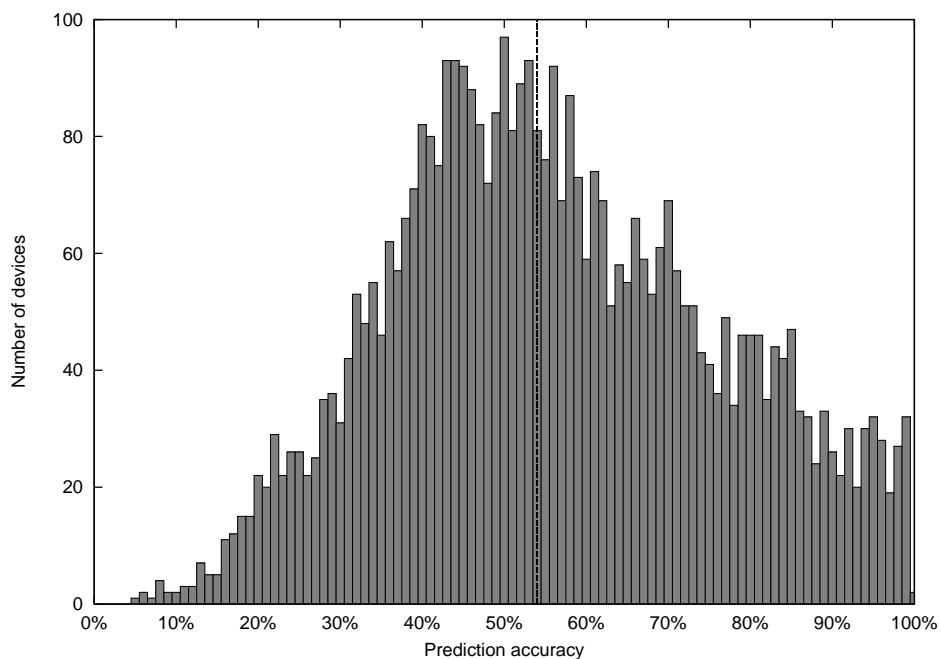


Figure 7.6: Accuracy of the Markov predictor for each client across all traces with more than 100 transitions. Dotted line denotes the average accuracy.

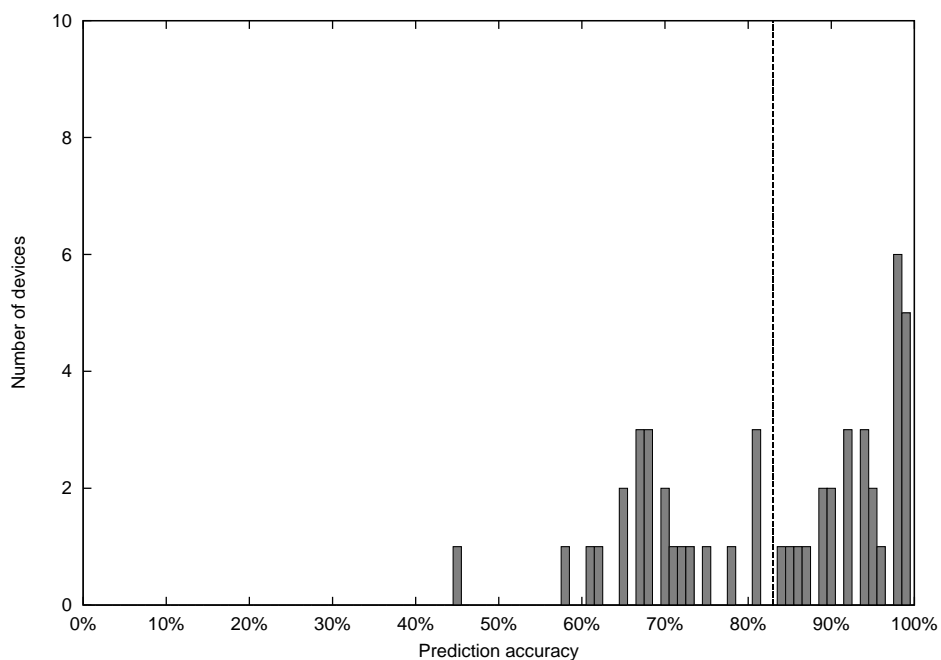


Figure 7.7: Accuracy of the Markov predictor for each of the top fifty client traces. Dotted line denotes the average accuracy.

7.2. Predict-on-disconnection

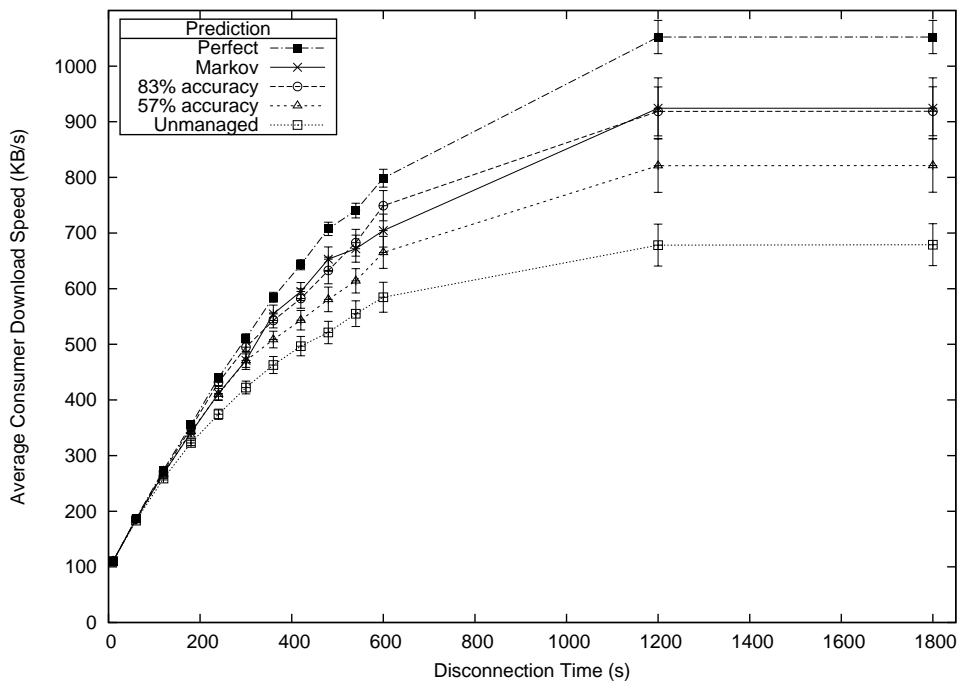


Figure 7.8: Impact of Markov prediction on the average consumer download speed, for real-world movement traces.

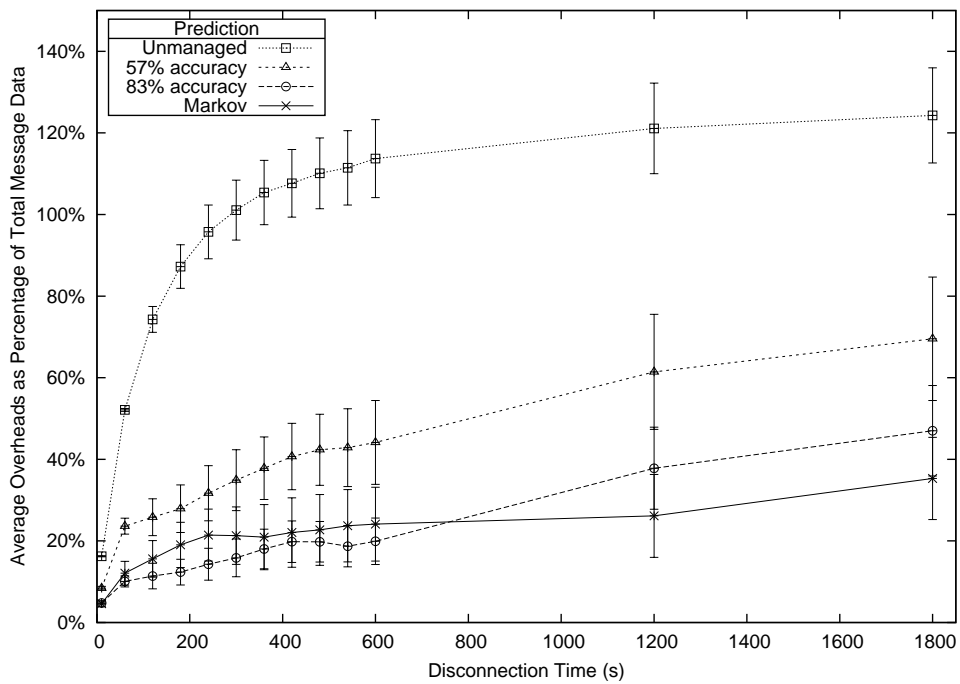


Figure 7.9: Impact of Markov prediction on average data overheads, for real-world movement traces.

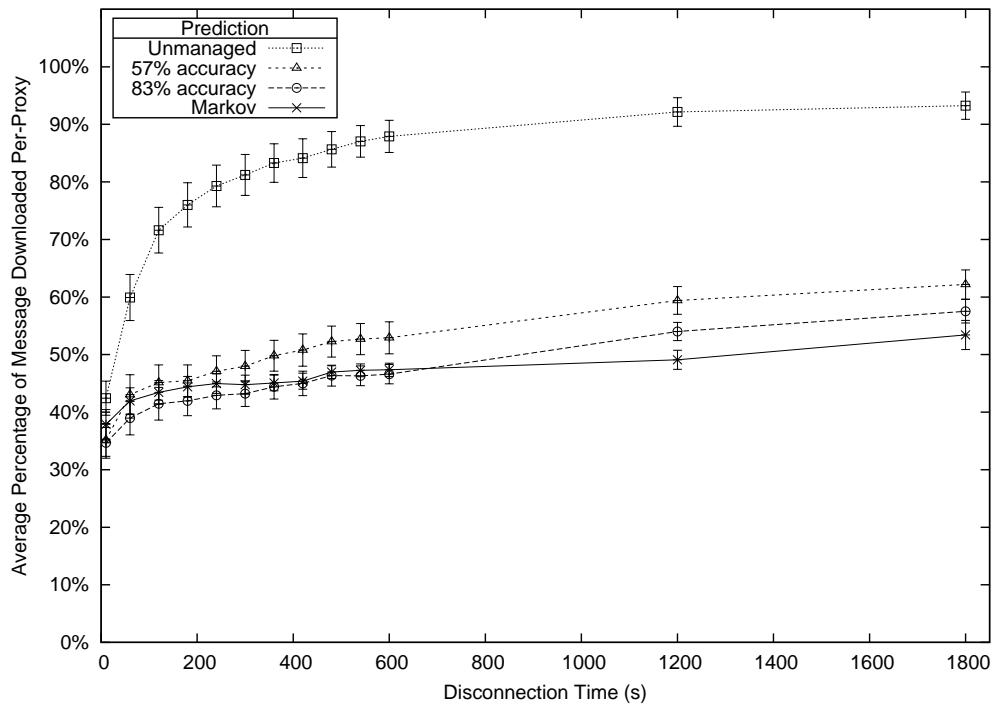


Figure 7.10: Average percentage of message downloads per-proxy, for real-world traces.

overall approximation reveals that Markov predictors can still be expected to yield noticeable improvement in download speed and data overheads over an unmanaged transfer, even at the lower average level of accuracy.

The overheads in Figure 7.9 remain quite high for the 57% accuracy predictor, but it appears that this is an unavoidable consequence of the pursuit of download speed. Directing a proxy to download pieces before a client arrives is a strategy that risks overheads for the sake of a large potential speed boost. Even if the prediction is incorrect, these additional piece sources help to boost overall speeds through parallel transfers. Costs are spread across multiple hosts, and no proxy will ever have to download more than the size of the message itself plus an insignificant control notification overhead. Figure 7.10 explores the average percentage of the message that each proxy downloads, calculated by dividing the total overhead by the number of unique proxies that the consumer contacts over the duration of the transfer. It shows that for the Markov and fixed accuracy predictors, each proxy only downloads about half of the message over the course of the transfer, which is a significant improvement over the unmanaged case. By reducing the burden on individual proxies, accurate prediction should reduce the cost of providing a proxy service, and therefore make it more practical for interested parties to do so.

The major drawback of using the Markov prediction algorithm is that it relies upon historical evidence. If a client is moving to a network that it has not previously visited, there

is no way for the client manager to predict this. If a client is truly following a migrating pattern, a client manager that uses a predict-on-disconnection MDP and the Markov algorithm is unable to provide any benefit during that period. Figure 5.21(a) presents the path lengths for the real-world client traces, measuring how many other connections a client makes between connections to a given access point. It shows that movements to new networks, represented by zero-length paths, are relatively rare and that clients instead tend to repeatedly revisit the same few networks. Movement to a previously unvisited proxy should therefore be infrequent enough that it does not significantly affect performance.

These results show that $O(2)$ Markov with fallback can be used to effectively predict future movements for most clients, and therefore increase consumer download speeds and reduce overheads compared to unmanaged Walkabout message transfers. This is especially true for clients with longer movement histories.

7.3 Supernode

If the predict-on-disconnection MDP uses an inaccurate algorithm, there is a good chance that data will be sent to the wrong proxy upon consumer disconnection, leading to transfer overheads. This bandwidth is not necessarily wasted, as it creates alternate sources of data. The correct proxy could draw upon these peers at a later point in time, and possibly increase its block download speed beyond what the source proxy's uplink could provide. However, there may be situations where these overheads are not considered to be worth the potential speed benefits, such as when home connections with monthly transfer quotas are involved. The supernode MDP offers a solution to this by restricting overheads to a single host that can afford them.

The supernode is a proxy that is guaranteed to have a fast connection to the Internet. It is the only proxy that downloads from the source proxy while the consumer is disconnected, so that it can provide blocks at a rate fast enough to saturate the new proxy's downlink when the consumer reconnects. This means that the consumer will experience the same moderately high download speeds regardless of its movement pattern, but that there is no chance for the accelerated download of locally cached pieces. Because non-supernode proxies only ever download when a consumer is present, their overheads will be almost non-existent.

The experiments that tested this MDP introduced a supernode to the network from Section 7.1.1, which was connected to the Internet node by a very fast 1GB/s link with 1ms latency (see Figure 7.11). When a proxy downloads blocks from the supernode, the maximum

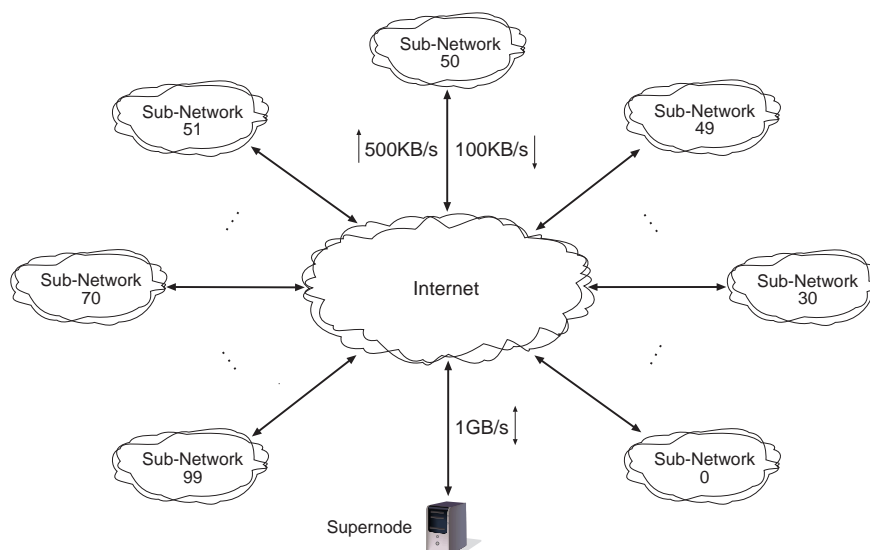


Figure 7.11: The supernode test network.

connection speed is restricted to 500KB/s by its downlink, so the request window size (RWS) of 2 used in previous experiments needed to be altered. Based upon Equation 5.1 and a conservative RTT of 200ms, the RWS was raised to 6. Consumers were compared when they were unmanaged, or managed using the supernode MDP or predict-on-disconnection MDP with a Markov prediction algorithm, under migrating and real-world movement patterns.

Figure 7.12 shows that the supernode policy generates a consumer download speed close to that of Markov prediction when disconnection periods are low, until it plateaus sharply. After that point, there is a small improvement in consumer download speed over what an unmanaged migrating consumer experiences, but it is slower than the others. There is a slight difference in download speed between the migrating consumer and its real-world counterpart under the supernode MDP, because the migrating consumer introduces a new proxy to the overlay each time it moves, which incurs a slight initialisation delay.

The advantage of the supernode MDP can be seen in the data overheads shown by Figure 7.13, which never exceed 100%. These levels are lower than for an unmanaged transfer, but remain higher than for Markov prediction. However, this cost is almost entirely borne by the supernode, which is selected because of its willingness to handle bulk data. The only overheads that the other proxies experience are due to control notifications and the blocks that are already en route when the consumer moves away. If the supernode's downloads are removed from the calculation, then the total overheads for any transfer it is involved with drop to nearly zero, which is far less than any of the alternatives. This remains true regardless of the consumer's movement, even for the worst-case migrating pattern.

7.3. Supernode

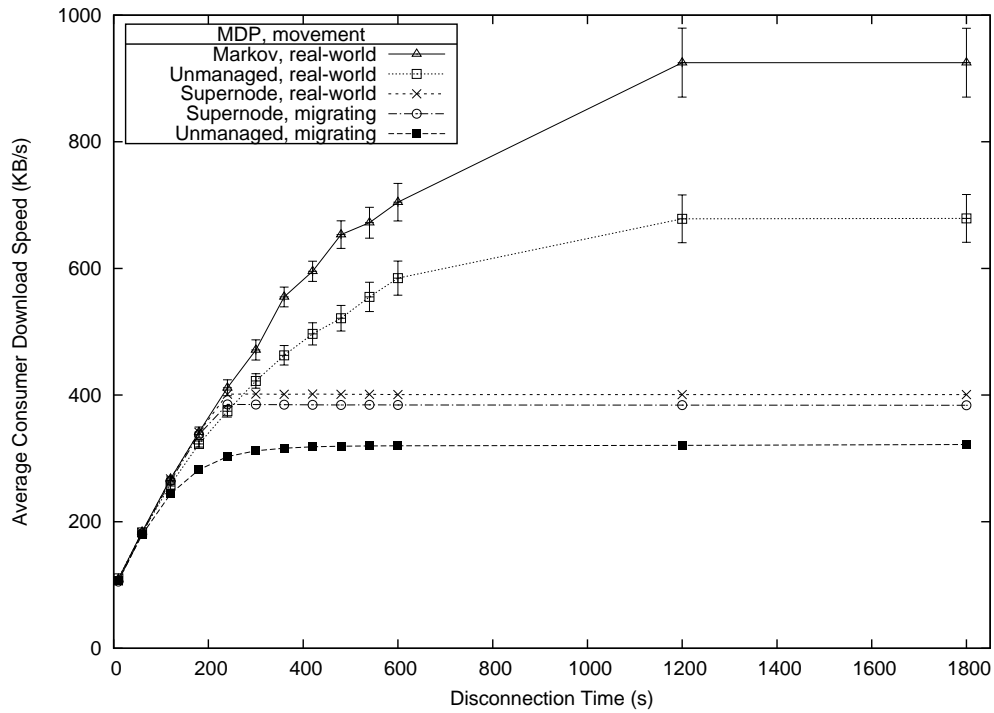


Figure 7.12: Impact of the supernode MDP on the average consumer download speed, for 500KB/s downlinks.

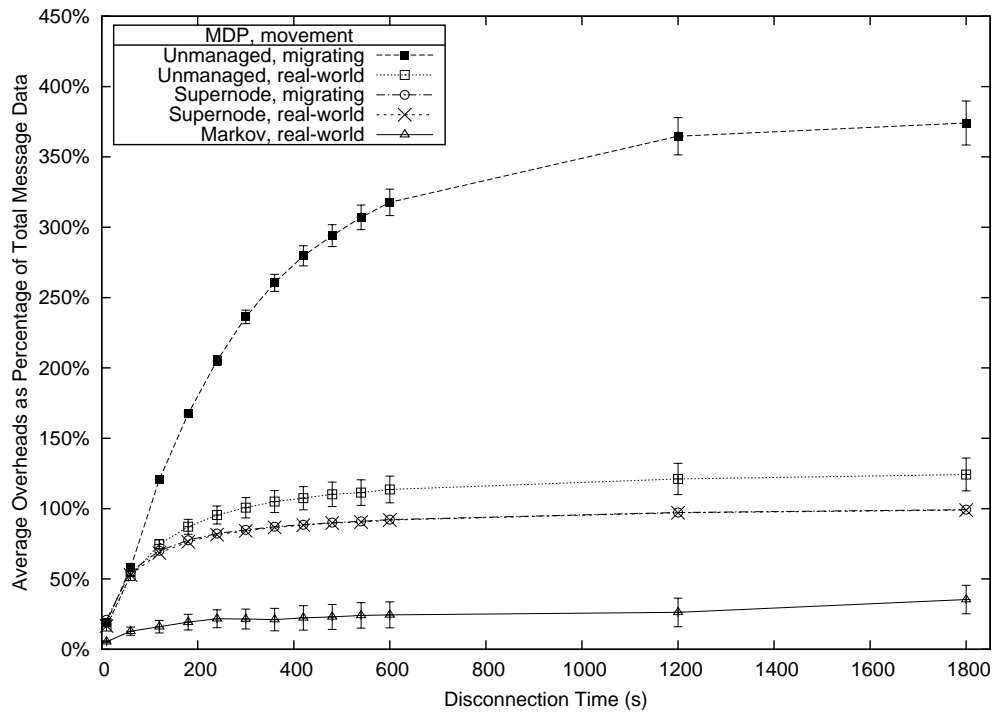


Figure 7.13: Impact of the supernode MDP on the average data overheads, for 500KB/s downlinks.

Table 7.3: Supporting variables for Equation 7.1.

Variable	Explanation
SN_m	Message data downloaded by the supernode during consumer disconnection
SN_u	Supernode uplink speed
S_u	Source proxy uplink speed
C_d	Current proxy downlink speed
C_m^{SN}	Message data downloaded from the supernode by the current proxy
t_c	Consumer connection time at the current proxy
t_d	Consumer disconnection time

The amount of data that accumulates at the supernode is a function of the consumer disconnection time. When this disconnection time is low, the proxy's fast downlink allows the consumer to retrieve all the data that the source previously uploaded to the supernode in its absence. A predict-on-disconnection scenario produces similar results under these circumstances, where the predicted proxy builds a cache of pieces in the consumers absence, and the consumer downloads them at local speed upon connection. The initial burst of speed is faster when the data is delivered in advance, but the average download speed is the same once all the previously uploaded data is exhausted, and Figure 7.12 confirms this.

Once the disconnection time rises beyond a critical point, which is 240 seconds in this experiment, the consumer is unable to retrieve all the cached data from the supernode. The proxy's downlink remains saturated for the duration of the transfer, which restricts the consumer download speed to that level regardless of the amount of data that accumulated at the supernode. By contrast, consumers with real-world movement patterns that are either unmanaged or managed with accurate prediction can benefit from larger caches of local data when they connect, and so their speeds continue to climb.

Finding a general formula for this turning point will reveal what the most favourable conditions are for the supernode MDP. Table 7.3 presents a summary of the variables that the following equations use. The key to finding this turning point is the amount of data C_m^{SN} that the consumer's current proxy is able to download from the supernode, and thus forward to the consumer, while the consumer is connected. The Internet links at the proxies are asymmetric, with their downlink greater than their uplink, so it is assumed that $S_u \leq C_d \leq SN_u$. If it is assumed that there is always an adequate supply of data at the source proxy, then the current proxy will download some data from the source proxy, and all remaining downlink capacity will be filled by the supernode as long as its supply of data remains. Therefore the first limiting

7.3. Supernode

condition on C_m^{SN} is that the proxy can only download as much data from the supernode as the remaining capacity allows:

$$C_m^{SN} \leq t_c(C_d - S_u)$$

The second limiting condition is that the proxy can only download as much data from the supernode as the supernode accumulated during the previous consumer disconnection period:

$$C_m^{SN} \leq SN_m = t_d S_u$$

The combined limit on the amount of data that the proxy can download from the supernode is:

$$C_m^{SN} \leq \min(t_c(C_d - S_u), SN_m)$$

The proxy's downlink remains saturated for the duration of the consumer connection when the available bandwidth is not enough to exhaust all of the data stored at the supernode:

$$C_m^{SN} \leq t_c(C_d - S_u) \leq SN_m$$

Therefore, the supernode MDP allows the proxy to make full use of its downlink when the following holds true for the duration of a message transfer:

$$\begin{aligned} t_c(C_d - S_u) &\leq SN_m = t_d S_u \\ C_d &\leq S_u \left(1 + \frac{t_d}{t_c}\right) \end{aligned} \quad (7.1)$$

This simplifies matters by not taking into account that the network properties and client connection patterns can vary over the course of the transfer. However, all of these properties are fixed in the test network. For this experiment, the inequality of Equation 7.1 holds true while $t_d \leq 240$ seconds, which corresponds exactly to the beginning of the plateau in Figure 7.12.

The supernode MDP aims to maximise the amount of data that a consumer can download while it is connected, by allowing it to download at a rate that is limited by the downlink of its current proxy, and not the uplink speeds of the other overlay peers. Equation 7.1 shows that once the downlink is saturated, it is not possible to improve the download speed any further by increasing the consumer disconnection time or increasing the source proxy uplink speed. However, an increase in proxy downlink speed can potentially yield improvements. In the case of these experiments, a higher downlink speed should allow consumer download speeds

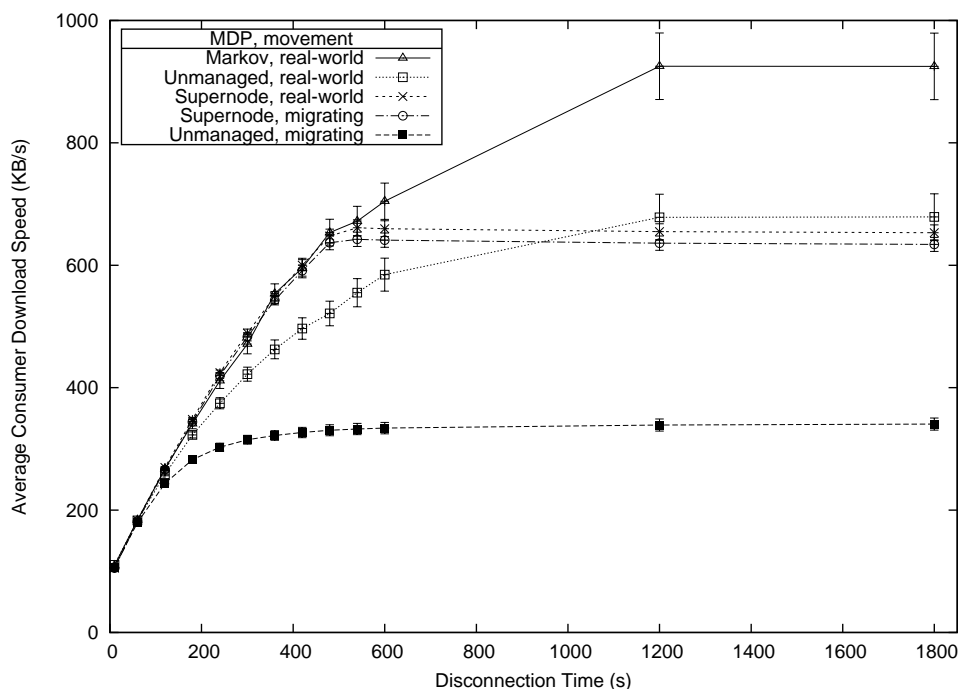


Figure 7.14: Impact of the supernode MDP on the average consumer download speed, for 1MB/s downlinks.

to increase as the disconnection time does, to flatten out at a higher value. A consumer that is unmanaged or managed by a predict-on-disconnection MDP will not experience the same improvements, because all of its downloads remain limited by the peer uplink speeds.

The next experiment repeated most of the conditions from the previous one, to test the effect of increasing the proxy downlink speed. Every proxy retained the 100KB/s uplink, but the downlink increased to 1MB/s and the RWS was raised to 12 accordingly. Equation 7.1 predicts that the supernode MDP should cause the consumer download speed to plateau at a disconnection time of 540 seconds, and Figure 7.14 confirms this. The supernode consumers experienced a significant jump in consumer download speeds at the longer disconnection periods when compared to Figure 7.12, rising at a rate equal to the highly accurate Markov prediction. They also outperformed the unmanaged real-world consumer when disconnection times were between 3 and 15 minutes. Even after the speeds levelled out, the average supernode and unmanaged speeds remained close. There was no increase in performance for the non-supernode methods, as they remained constrained by the proxy uplink speeds. Thus when proxy downlink speeds are significantly faster than uplink speeds, the supernode MDP becomes a viable alternative to a predict-on-disconnection MDP.

These results show that the supernode MDP does have a role. It is a low risk option, as all consumer movement patterns produce very similar consumer download speeds, and

overheads are always minimised for the non-supernode proxies. The low overheads may be particularly important if a consumer's path is hard to predict, and a predict-on-disconnection MDP would generate large amounts of excess traffic through poor proxy selection. As with an accurate predict-on-disconnection MDP, its performance improves when disconnection times are longer, because this increases the amount of stored data that is available for the consumer to download at high speed when it connects. Performance of the supernode MDP improves significantly as the downlink speeds increase in relation to uplink speeds, and can equal the Markov prediction-on-disconnection MDP, regardless of how predictable the client movement is.

7.4 Summary

Chapter 5 shows that Walkabout makes Internet transfers practical when they involve mobile producers or mobile consumers with repetitive movements. This chapter demonstrates that the use of client managers to organise pre-emptive message delivery improves consumer download speeds, and thus effective transfer speeds, when consumers are mobile. This includes the scenario where a highly mobile consumer visits many different locations, which was impractical under the original transfer model. These findings complete Contribution **C3** from Section 1.4.

A client manager using the predict-on-disconnection MDP and a perfect prediction algorithm can deliver high download speeds to even the most mobile consumers, equalling those that an unmanaged consumer with repetitive movements experiences. Therefore, an algorithm that delivers perfect prediction makes all Walkabout transfers practical, regardless of the consumer movement patterns. A Markov prediction algorithm provides high speeds as a result of its accuracy, but it is only able to predict future movements based upon past movements, making it unsuitable for a migrating consumer. Users can deploy their own specialised prediction algorithms if a general approach does not produce the desired performance levels, and Chapter 8 explores some possibilities. Even low accuracy prediction can deliver higher speeds than are possible for an unmanaged transfer.

If movement is completely unpredictable, or overheads must be kept to a minimum, then the supernode MDP is an appropriate alternative. It provides consistent consumer download speeds regardless of the consumer's movement pattern, and speeds are close to those for accurate prediction-based methods when the proxy downlink bandwidth is high in comparison to the uplink bandwidth of the other overlay peers. Overheads for the supernode never exceed 100%, and the overheads at the other overlay proxies are negligible.

The results in this chapter prove that Walkabout can make it practical to deliver large messages to any mobile consumer, regardless of its movement patterns. In combination with the results from Chapter 5, these results show that Walkabout makes transfers involving *any* mobile devices practical, and thus satisfies the aims identified in the introduction to this thesis.

Chapter 8

Conclusions and future directions

This thesis presented the design for Walkabout, an architecture that uses local proxies to provide asynchronous Internet messaging support to mobile devices. The use of local proxies was shown to enable rapid local uploads and downloads, which make it practical for mobile devices to send and receive arbitrarily large messages, regardless of their mobility patterns. End-to-end transfer speeds were found to be high, reaching levels that were close or equal to those achieved between fixed devices in many situations (and even exceeding them in some).

Walkabout enables a wide variety of mobile applications, that were previously rendered impractical by the difficulties associated with transferring large data objects over the Internet. Chapter 2 presented some examples of these, including applications for media backup, distribution, retrieval and messaging. The architecture is easy to deploy on existing networks using common hardware, and many existing devices have the necessary wireless networking capabilities to access it.

8.1 Contributions

This thesis has advanced the state of the art in the field of mobile networking by way of three major contributions.

Contribution **C1** presents the literature that is related to this work. Chapter 3 explores the strengths and weaknesses of existing research when applied to Internet data transfers involving mobile devices. It covers such areas as service discovery, the use of proxies, peer-to-peer overlay networks, and how to locate, maintain communication with, and predict the movement patterns of mobile devices. The review shows that Walkabout shares some similarities with existing research such as Infostations and Hagggle, but that it is unique as a complete system.

Contribution **C2** covers the core design of the Walkabout architecture. Chapter 4 described the architecture, which provides a store-and-forward message delivery service that enables mobile devices to take advantage of local networking speeds during their potentially limited periods of connectivity. Chapter 5 tested this architecture through simulation, found that it scales well to large files and multiple consumers, and that it makes mobile data transfers practical in some situations. A mobile producer is always able to upload rapidly at local speed, and the cache of data that accumulates at the proxy enables the Internet transfer to continue at high speed, even if the producer disconnects. If the ratio of connection to disconnection time is high enough, the effective transfer speeds are equal to those generated by a fixed producer. If the producer is moving between networks rapidly, the creation of multiple data caches may increase the effective uplink bandwidth to the point where transfer speeds actually exceed what is possible for a fixed producer. Consumer download speeds are limited by the supply of data that the proxy accumulates. If the consumer is constantly reconnecting to the same network, the proxy accumulates data in its absence, and the consumer is able to download the stored data at local speed upon reconnection. This too can generate high effective transfer speeds that are unaffected by mobility. Speeds are lower, though still good, if a consumer only visits a small number of networks in a repetitive movement pattern.

Contribution **C3** covers the client manager extension to Walkabout. Chapter 6 described this extension, where a host coordinates the data retrieval actions of the proxies in an effort to pre-emptively deliver messages to the best location before a consumer connects. Chapter 7 simulated the performance of the client manager, which depends upon the configuration of a message distribution policy (MDP) and prediction algorithm. The results showed that the addition of a client manager makes mobile data transfers practical in the remaining situations where the original design did not. A predict-on-disconnection MDP with a perfect prediction algorithm provides any mobile consumer with high download speeds upon connection, generating end-to-end transfer speeds that are close to what a fixed consumer would experience. The Markov prediction algorithm was shown to be a practical option, with enough accuracy to deliver high speeds for consumers with movement based upon real-world traces. A supernode MDP offers moderate transfer speeds that are consistent regardless of the consumer's movement patterns, approaching those of accurate predict-on-disconnection distribution as proxy downlink bandwidth increases in comparison to uplink bandwidth.

Existing methods for transferring large amounts of data across the Internet are impractical when mobile devices are involved. The central hypothesis of this research is that asynchronous messaging supported by local proxies can make these transfers practical. Together,

the contributions made by this thesis show that the Walkabout architecture uses this approach to make Internet transfers practical for mobile devices, and therefore that the hypothesis is valid.

8.2 Future directions

While this thesis covers the complete design and evaluation of Walkabout, it still leaves scope for enhancements. This section presents some of the directions that future research could take, covering proxy message scheduling, support for streaming media, media transcoding, and alternative client manager configurations. Where appropriate, it also offers a basic outline of how these changes could be incorporated in to the existing architecture. The section concludes by briefly summarising the remaining future research possibilities that were suggested within this thesis.

8.2.1 Message scheduling

As described in Chapter 4, the standard action that a proxy takes when it receives a new header is to join (or create) the message overlay, then exchange blocks with its peers until all consumers have the message. This can become unmanageable for a proxy in a busy environment, such as an office or a public access point. If the proxy receives a high volume of headers, and thus participates in a large number of overlays simultaneously, each transfer will only have access to a small share of the available bandwidth. A better approach is for the proxy to concentrate on maximising the performance of just a few transfers at a time. Therefore, an avenue for future research is to institute message scheduling on proxies and evaluate the performance of different scheduling algorithms.

A proxy running a message scheduler would not join the overlay as soon as it receives a new header, but would instead add that header to a priority queue. The scheduling algorithm would determine the queuing priority, and dictate when the proxy should take a header from the front of the queue to join an overlay, or leave an overlay and return the header to the end of the queue. The queuing priority might be based upon the order that headers are received in, who the message is addressed to or sent by, or the size of the message payload.

The simplest scheduling approach is for the proxy to join a small number of overlays and focus solely on them until the message delivery is complete. This would include some where the proxy is both downloading and uploading blocks, and others that are only uploads. Each time a message transfer completes, the scheduler could take the next header from the front of the queue and join the new overlay. Alternatively, the scheduler might only allocate a

limited amount of time to each overlay, so that the proxy leaves the overlay and reintroduces the header to the queue if the period is not long enough to complete the transfer.

Any future research in this area would need to determine the circumstances under which different scheduling algorithms and parameters are effective. The different approaches should aim to give all local users equal access to the service, while considering the effect that fluctuating overlay membership could have on global message delivery performance.

8.2.2 Support for streaming media

Message delivery is atomic from the perspective of Walkabout client applications. A consumer downloads any pieces that are available at the proxies it connects to, but only makes the message available to local applications once it has all of the pieces. It does not matter what order the consumer downloads these pieces in, so the proxies are able to select blocks, and thus pieces, from their peers in the order that minimises total message delivery time. A consequence is that Walkabout does not support streaming media, where the consumer plays the contents of a media message as pieces arrive, rather than waiting for the entire message. A possible research direction is therefore to adapt Walkabout's message delivery so it does support streaming media.

There are two types of streaming media to consider. A producer might stream the contents of an existing media file, and this should only require minor modifications to Walkabout's delivery mechanisms. Streaming a continuous and potentially infinite stream of data, such as live television or radio, would require more significant changes.

It may be possible to support the streaming of stored media within regular Walkabout messages by making two changes made to the delivery process. The first is that proxies need to modify their block selection algorithm, so that they retrieve blocks in sequential piece order from their overlay peers. This change in policy could be triggered by the producer flagging that delivery is streaming within the message header. The other change is that the consumer needs to make pieces available to the media player application as they arrive, rather than waiting for the entire message. If the application is able to build a big enough buffer of pieces each time the device is connected, it may be possible to present the user with uninterrupted playback.

The sequential block selection algorithm is likely to provide slower overlay transfer speeds than the original, but this does not present a major problem. While a consumer is connected, the speed only needs to be fast enough to match the playback bitrate. The consumer can use any additional bandwidth to buffer pieces for playback during disconnection. Maintaining uninterrupted playback may also require support from the proxy's scheduling algorithm,

to provide a minimum quality-of-service level by prioritising streaming messages over those more tolerant of delay. Future research could determine the best way to provide this streaming support, evaluate how much difference the modified block selection algorithm makes to transfer speeds in comparison to non-streaming transfers, and find what conditions allow for uninterrupted playback.

Adapting Walkabout to support continuous streaming content would be more difficult. While it could potentially be achieved by sending a series of finite streaming messages, or allowing the creation of overlays without prior knowledge of the entire message, there would be little benefit in adding this support. Walkabout is aimed at devices that have limited, intermittent connectivity, which are able to carry out bursts of asynchronous communication while connected. Continuous streaming media is a synchronous application, because applications consume data at the same rate it is created and there is no way to build a buffer without incurring a significant playback delay. If a consumer device's connection patterns are consistent enough for it to play a continuous stream, then it does not require the support of Walkabout and could easily use some other protocol.

8.2.3 Media transcoding

When a mobile device downloads a media file over Walkabout, it may find that it is unable to experience the full level of detail present within the original file. This would typically be because its screen resolution is lower than that of a video or image, but could also be related to other factors like media bitrate, colour depth, or encoding format. A less detailed version of the file would be sufficient for the device, and would probably take less time to transfer across the Internet. Section 3.4 showed how proxies have been used to transcode media and improve delivery times to mobile devices, and this is something that may be worth introducing to Walkabout in these situations.

The Internet is the main bottleneck in a Walkabout transfer, so either the producer or the proxy that it uploads its pieces to should be responsible for adapting the media. The simplest option is for the producer to do so, but shifting the task to the proxy does have advantages. If the producer is a mobile device, then the proxy is probably more capable of carrying out the intensive computations associated with transcoding. The other advantage is that the producer does not need to have any knowledge of the capabilities of the consumer, or indeed that any transcoding is being performed. It can simply upload the file as per normal, and let the proxy handle the rest.

If the proxy is responsible for transcoding, it can determine the best format through communication with a user's identity manager. An identity manager normally forwards message headers to the appropriate consumers transparently. If the message contains media and the recipient is a limited device, the identity manager could instead reply to the proxy, to request that it transforms the media to a new resolution or format before delivery proceeds. If the consumer does not have an identity manager, it could send its own transcode request back to the source proxy upon receipt of the message header.

Transcoding a media message on the proxy has several potential issues. A proxy initiating a regular Walkabout transfer can create the overlay and contact the consumers as soon as it receives the header from the producer. However, a proxy that is going to transcode a message needs to wait until it has all the pieces and has completed the transformation before the delivery can start. This could lead to a significant transfer delay, even if the resultant message is smaller than the original. A transcoded message will have a different header to the original and thus require a separate delivery overlay, which will affect the efficiency of a transfer to multiple consumers, particularly if each one requests a different format. There is also the problem that if the producer is moving, a single proxy may not receive all of the pieces, and so one of them would need to download the message in full from its peers before the transformation can occur.

Even with these problems, transcoding could deliver potential speed benefits. If the message is smaller, it will take less time to transfer across the Internet and forward to the consumer. The speed gains may be enough to offset the time lost during the initial transformation, yet the output on the device may be indistinguishable from what the original file would yield.

8.2.4 Client manager modules

Chapter 7 showed that use of a client manager to coordinate pre-emptive delivery can lead to high transfer speeds for mobile consumers. How high these speeds are depends upon the MDP and prediction algorithm that the client manager is using. One area of potential research is to produce new MDPs using different prediction algorithms, and evaluate how they can help to improve transfer speeds in particular situations.

The predict-on-disconnection MDP tested in Chapter 7 has the potential to generate high transfer speeds to mobile consumers, but the actual performance is governed by how accurate the prediction algorithm is. Markov prediction analyses a device's historical movement sequences and was found to be effective when consumers have repetitive movement patterns. It does not work well if their movements do not possess any underlying patterns, and it is unable to determine when an otherwise predictable consumer is going to make a movement

that is different to their regular patterns. Therefore Walkabout could benefit from prediction algorithms that use other information in addition to or in place of movement history analysis.

Cellular networking could help to improve prediction in various ways. Knowledge of mobile phone cell location could provide the prediction algorithm with a coarse-grained tracking mechanism, or the client could use the cellular network to send regular GPS coordinate updates to the client manager. Direct user input to inform the system of their movement plans could yield great improvements, particularly if they are heading to a location that is not part of their regular patterns. This would quickly become annoying for the user if they had to provide input constantly, but it could provide valuable improvements in otherwise unpredictable scenarios if used sparingly. Calendar integration could also be a less intrusive way to get direct user input on their likely movements. If multiple context sources are available, then a user modelling service could consider them all, resolve any conflicting information, and provide the most likely location to the MDP.

8.2.5 Other possibilities

The following are some remaining possibilities for future research that were raised during the course of this thesis:

- The formulation and evaluation of different proxy cache management policies (Section 4.4).
- Creating a framework for the development of trust relationships between proxies (Section 4.7).
- Coordinating block downloads between peers, in order to reduce the stress on the source peer and therefore improve overall transfer speeds (Section 5.2.2).
- Evaluating the improvements in transfer speeds that could result from a producer repeatedly uploading pieces of a message to the different proxies it visits (Section 5.3.3).

References

- ABERER, K. & DESPOTOVIC, Z. (2001). Managing trust in a peer-2-peer information system. In *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM 2001)*, 310–317, Atlanta, Georgia, USA.
- ADYA, A., BOLOSKY, W.J., CASTRO, M., CERMAK, G., CHAIKEN, R., DOUCEUR, J.R., HOWELL, J., LORCH, J.R., THEIMER, M. & WATTENHOFER, R.P. (2002). Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, 1–14, Boston, Massachusetts, USA.
- AKOUSH, S. & SAMEH, A. (2007). Mobile user movement prediction using bayesian learning for neural networks. In *Proceedings of the 2007 International Conference on Wireless Communications and Mobile Computing (IWCMC 2007)*, 191–196, Honolulu, Hawaii, USA.
- APPLE COMPUTER, INC. (2006). Bonjour overview. Retrieved 20 June 2007 from <http://developer.apple.com/documentation/Networking/Bonjour-date.html>.
- APPLE COMPUTER, INC. (2007). Apple - MacBook Pro. Retrieved 17 July 2007 from <http://www.apple.com/macbookpro/>.
- ASSAD, M., KAY, J. & KUMMERFELD, B. (2005). The Keep-in-Touch system. In *Online Proceedings of the UbiComp 2005 Workshop on Situating Ubiquitous Computing in Everyday Life: Bridging the Social and Technical Divide*, Tokyo, Japan.
- AZUREUS (2007). Azureus: Java BitTorrent client. retrieved 27 July 2007 from <http://azureus.sourceforge.net>.
- BABENHAUSERHEIDE, A. (2004). Gnutella for users. Retrieved 20 June 2007 from <http://draketo.de/inhalt/krude-ideen/gnufu-en.pdf>.
- BALAKRISHNAN, H., LAKSHMINARAYANAN, K., RATNASAMY, S., SHENKER, S., STOICA, I. & WALFISH, M. (2004). A layered naming architecture for the internet. *SIGCOMM Computer Communications Review*, **34**, 343–352.
- BBC NEWS (2007). Two cautioned over wi-fi ‘theft’. Retrieved 27 July 2007 from <http://news.bbc.co.uk/1/hi/england/hereford/worcs/6565079.stm>.
- BECK, J., GEFFLAUT, A. & ISLAM, N. (1999). MOCA: A service framework for mobile computing devices. In *Proceedings of the 1st ACM international workshop on Data engineering for wireless and mobile access (MobiDE)*, 62–68, Seattle, Washington, USA.

- BHATTACHARYA, A. & DAS, S.K. (2002). LeZi-update: An information-theoretic framework for personal mobility tracking in PCS networks. *Wireless Networks*, **8**, 121–135.
- BLUETOOTH SIG (2003). *Bluetooth Specification Version 1.2*, chap. Service Discovery Protocol (SDP), 112–170.
- BOINGO WIRELESS (2007). Wireless service at Boingo. Retrieved 27 July 2007 from <http://www.boingo.com>.
- BRAMPTON, A., MACQUIRE, A., RAI, I.A., RACE, N.J.P. & MATHY, L. (2006). Stealth distributed hash table: A robust and flexible super-peered DHT. In *Proceedings of the 2nd Conference on Future Networking Technologies (CoNEXT'06)*, 216–227, Lisboa, Portugal.
- BUCHHOLZ, S. & SCHILL, A. (2002). Web caching in a pervasive computing world. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, 14–18, Orlando, Florida, USA.
- BURCEA, I., JACOBSEN, H.A., DE LARA, E., MUTHUSAMY, V. & PETROVIC, M. (2004). Disconnected operation in publish/subscribe middleware. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)*, 39–50, Berkeley, California, USA.
- BURGESS, J., GALLAGHER, B., JENSEN, D. & LEVINE, B.N. (2006). MaxProp: Routing for vehicle-based disruption-tolerant networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, 1688–1698, Barcelona, Spain.
- BUSH, J., IRVINE, J. & DUNLOP, J. (2005). Multimedia content delivery using infostations. In *Proceedings of the 62nd IEEE Vehicular Technology Conference*, vol. 4, 2191–2195, Dallas, Texas, USA.
- CAPORUSCIO, M., CARZANIGA, A. & WOLF, A.L. (2003). Design and evaluation of a support service for mobile, wireless publish/subscribe applications. Tech. Rep. CU-CS-944-03, Department of Computer Science, University of Colorado.
- CARZANIGA, A., ROSENBLUM, D.S. & WOLF, A.L. (1998). Design of a scalable event notification service: Interface and architecture. Tech. Rep. CU-CS-863-98, Department of Computer Science, University of Colorado.
- CASTRO, M., DRUSCHEL, P., KERMARREC, A.M. & ROWSTRON, A. (2002). SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, **20**, 1489–1499.
- CHENG, C., JAIN, R. & VAN DEN BERG, E. (2003). *Wireless internet handbook: technologies, standards, and applications*, chap. Location prediction algorithms for mobile wireless systems, 245–263. CRC Press, Inc., Boca Raton, Florida, USA.
- CHESHIRE, S. & KROCHMAL, M. (2006a). DNS-based service discovery. IETF draft, work in progress.

- CHESHIRE, S. & KROCHMAL, M. (2006b). Multicast DNS. IETF draft, work in progress.
- COHEN, B. (2003). Incentives build robustness in BitTorrent. In *Online Proceedings of the 1st Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, California.
- COHEN, B. (2005). Bittorrent goes trackerless: Publishing with BitTorrent gets easier! Retrieved 8 May 2006 from <http://www.bittorrent.com/trackerless.html>.
- CUGOLA, G., DI NOTTO, E. & FUGGETTA, A. (2001). The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, **27**, 827–850.
- CZERWINSKI, S.E., ZHAO, B.Y., HODES, T., JOSEPH, A.D. & KATZ, R. (1999). An architecture for a secure service discovery service. In *Proceedings of the 5th Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, 24–35, Seattle, Washington, USA.
- DABEK, F., KAASHOEK, M.F., KARGER, D., MORRIS, R. & STOICA, I. (2001). Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, 202–215, Banff, Canada.
- DABEK, F., ZHAO, B., DRUSCHEL, P., KUBIATOWICZ, J. & STOICA, I. (2003). Towards a common API for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 33–44, Berkeley, California, USA.
- DAVIS, J.A., FAGG, A.H. & LEVINE, B.N. (2001). Wearable computers as packet transport mechanisms in highly-partitioned ad-hoc networks. In *Proceedings of the 5th IEEE International Symposium on Wearable Computers (ISWC 2001)*, 141–148, Zurich, Switzerland.
- EGGERT, L., LAGANIER, J., LIEBSCH, M. & STIEMERLING, M. (2004). HIP resolution and rendezvous mechanisms. In *Online Proceedings of 1st Workshop on HIP and Related Architectures*, Washington DC, USA.
- EL-ANSARY, S. & HARIDI, S. (2005). *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*, chap. An Overview of Structured Overlay Networks, 665–684. Auerbach Publications, Boca Raton, Florida, USA.
- ELLIS, L. (2006). Bittorrent's swarms have a deadly bite on broadband nets. retrieved 27 July 2007 from <http://www.multichannel.com/article/CA6332098.html>.
- EUGSTER, P.T., FELBER, P.A., GUERRAQUI, R. & KERMARREC, A.M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, **35**, 114–131.
- FORD, B. (2004). Unmanaged internet protocol: taming the edge network management crisis. *SIGCOMM Computer Communications Review*, **34**, 93–98.
- FRENKIEL, R.H. & IMIELINSKI, T. (1996). Infostations: The joy of “many-time, many-where” communications. Tech. Rep. TR-119, WINLAB, Rutgers University.
- FRENKIEL, R.H., BADRINATH, B.R., BORRÀS, J. & YATES, R.D. (2000). The infostations challenge: Balancing cost and ubiquity in delivering wireless data. *IEEE Personal Communications*, **7**, 66–71.

References

- GEDC (2007). GEDC researchers achieve 15Gbps wireless transmission speeds. Retrieved 27 July 2007 from <http://www.gedcenter.org/news20.html>.
- GELERNTER, D. (1985). Generative communication in Linda. *ACM Computing Surveys*, **7**, 80–112.
- GONG, L. (2002). Project JXTA: A technology overview. Retrieved 20 June 2007 from http://www.jxta.org/project/www/docs/jxtaview_01nov02.pdf.
- GOODMAN, D.J., BORRÀS, J., MANDAYAM, N.B. & YATES, R.D. (1997). Infostations: A new system model for data and messaging services. In *Proceedings of the 47th IEEE Vehicular Technology Conference*, vol. 2, 969–973.
- GOOGLE (2007a). Blogger: Create your blog now. Retrieved 9 July 2007 from <http://www.blogger.com>.
- GOOGLE (2007b). Blogger help: On the go with blogger mobile. Retrieved 9 July 2007 from <http://help.blogger.com/bin/answer.py?answer=42448>.
- GOOGLE (2007c). Google. Retrieved 9 July 2007 from <http://www.google.com>.
- GOOGLE (2007d). Google WiFi Mountain View coverage map. Retrieved 4 July 2007 from <http://wifi.google.com/city/mv/apmap.html>.
- GOOGLE (2007e). YouTube - broadcast yourself. Retrieved 9 July 2007 from <http://www.youtube.com>.
- GOOGLE (2007f). YouTube - broadcast yourself - how do I upload a video from my mobile device to YouTube? Retrieved 9 July 2007 from <http://www.google.com/support/youtube/bin/answer.py?answer=70029&topic=11840>.
- GROTH, D. (2006). NodeDB.com - the wireless database project. Retrieved 27 July 2007 from <http://www.nodedb.com/index.php>.
- GUTTMAN, E. (1999). Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing*, **3**, 71–80.
- GUTTMAN, E., PERKINS, C., VEIZADES, J. & DAY, M. (1999). RFC 2608 - Service location protocol, version 2.
- HSIAO, H.C. & KING, C.T. (2005). Mobility churn in DHTs. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems and Workshops (ICDCSW'05)*, 799–805, Washington DC, USA.
- HSUPA.COM (2007). Vodafone Germany launches HSUPA data card. Retrieved 17 July 2007 from http://www.hsupa.com/index.php?option=com_content&task=view&id=49&Itemid=31.
- IACONO, A.L. & ROSE, C. (2000). *Next Generation Wireless Networks*, chap. Infostations: New Perspectives on Wireless Data Networks, 1–60. Kluwer Academic Publishers.

- IYER, S., ROWSTRON, A. & DRUSCHEL, P. (2002). SQUIRREL: A decentralized, peer-to-peer web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, 213–222, Monterey, California, USA.
- IZAL, M., URVOY-KELLER, G., BIRSACK, E., FELBER, P., HAMRA, A. & GARCES-ERICE, L. (2004). Dissecting BitTorrent: Five months in a torrent’s lifetime. In *Proceedings of the 5th International Workshop on Passive and Active Measurement (PAM 2004)*, 1–11, Antibes Juan-les-Pins, France.
- JOSHI, A., WEERAWARANA, S. & HOUSTIS, E.N. (1997). On disconnected browsing of distributed information. In *Proceedings of the 7th IEEE International Workshop on Research Issues in Data Engineering (RIDE’97)*, 101–107, Birmingham, UK.
- JUANG, P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L.S. & RUBENSTEIN, D. (2002). Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 96–107, San Jose, California, USA.
- KARA, H. & EDWARDS, C. (2003). A caching architecture for content delivery to mobile devices. In *Proceedings of the 29th EUROMICRO Conference “New Waves in System Architectures” (EUROMICRO’03)*, 241–248, Antalya, Turkey.
- KISTLER, J. & SATYANARAYANAN, M. (1992). Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, **10**, 3–25.
- KODAK (2005). Kodak EasyShare-One zoom digital camera: User’s guide. Retrieved 9 July 2007 from http://www.kodak.com/global/plugins/acrobat/en/service/manuals/urg00569/EasyShare-One_GLB_en.pdf.
- KOTZ, D., HENDERSON, T. & ABYZOV, I. (2005). CRAWDAD trace set dartmouth/campus/movement (v. 2005-03-08). Retrieved from <http://crawdad.cs.dartmouth.edu/dartmouth/campus/movement>.
- KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C. & ZHAO, B. (2000). Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, 190–201, ACM.
- LANDFELDT, B., HASSAN, J., ZOMAYA, A., MANITPORNUT, S. & SUBRATA, R. (2006). Titan: A new paradigm in wireless internet access based on community collaboration. In *Proceedings of the International Conference on Wireless Communications and Mobile Computing (IWCMC 2006)*, 331–336, Vancouver, Canada.
- LI, J., STRIBLING, J., GIL, T.M., MORRIS, R. & KAASHOEK, M.F. (2004). Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS’04)*, 87–99, San Diego, California, USA.
- LIANG, J., KUMAR, R. & ROSS, K.W. (2006). The FastTrack overlay: A measurement study. *Computer Networks*, **50**, 842–858.

- LIU, G.Y. & MAGUIRE, G.Q. (1995). Efficient mobility management support for wireless data services. In *Proceedings of the 45th IEEE Vehicular Technology Conference*, vol. 2, 902–906, Chicago, Illinois, USA.
- MALAMUD, C. & ROSE, M. (1993). RFC 1528 - principles of operation for the TPC.INT subdomain: Remote printing - technical procedures.
- MANTARA SOFTWARE (2003). Elvin subscription language reference v4.0.0. retrieved 27 July 2007 from <http://www.mantara.com/support/docs/guides/elvin-sublang/elvin-sublang-4.0.pdf>.
- MAYMOUNKOV, P. & MAZIERES, D. (2002). Kademia: a peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 53–65, Cambridge, Massachusetts, USA.
- MOCKAPETRIS, P. (1997a). RFC 1034 - Domain names - concepts and facilities.
- MOCKAPETRIS, P. (1997b). RFC 1035 - Domain names - implementation and specification.
- MOSKOWITZ, R. & NIKANDER, P. (2006). RFC 4423 - Host identity protocol (HIP) architecture.
- MOSKOWITZ, R., NIKANDER, P., JOKELA, P. & HENDERSON, T. (2007). Host identity protocol. IETF draft, work in progress.
- MUTHITACHAROEN, A., MORRIS, R., GIL, T.M. & CHEN, B. (2002). Ivy: A read/write peer-to-peer file system. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, 31–44, Boston, Massachusetts, USA.
- NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (2001). Specification for the advanced encryption standard (AES). Federal Information Processing Standards Publication 197, retrieved 17 July 2007 from <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- NIKANDER, P., YLITALO, J. & WALL, J. (2003). Integrating security, mobility, and multi-homing in a HIP way. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'03)*, 87–99, San Diego, California, USA.
- OECD (2007). OECD broadband portal. Retrieved 17 March 2008 from <http://www.oecd.org/sti/ict/broadband>.
- OPEN MOBILE ALLIANCE (2005). OMA enabler releases and specifications - OMA multimedia messaging service. Retrieved 9 July 2007 from http://www.openmobilealliance.org/release_program/mms_v1_3.html.
- PELUSI, L., PASSARELLA, A. & CONTI, M. (2006). Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Communications Magazine*, **44**, 134–141.
- PENTLAND, A., FLETCHER, R. & HASSON, A. (2004). DakNet: Rethinking connectivity in developing nations. *IEEE Computer*, **37**, 78–83.
- PERKINS, C. (2002). RFC 3344 - IP mobility support for IPv4.

- PERKINS, C. & JOHNSON, D. (2000). Route optimization in mobile IP. IETF draft, work in progress.
- POUWELSE, J.A., GARBACKI, P., EPEMA, D.H.J. & SIPS, H.J. (2005). The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, 205–216, Ithaca, NY, USA.
- PRESS ESC (2007). Multi-gigabit wireless “within three years”. Retrieved 27 July 2007 from http://pressesc.com/01184858412_multi_gigabit_wireless.
- PYTHON (2007). Python programming language – official website. Retrieved 9 July 2007 from <http://www.python.org>.
- RAKOTONIRAINY, A. & GROVES, G. (2002). Resource discovery for pervasive environments. In *Proceedings of the 4th International Symposium on Distributed Objects and Applications (DOA 2002)*, 866–883, University of California, Irvine, USA.
- RAMASUBRAMANIAN, V. & SIRER, E.G. (2004). The design and implementation of a next generation name service for the internet. In *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'04)*, 331–342, Portland, Oregon, USA.
- RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R. & SHENKER, S. (2001). A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, 161–172, San Diego, California, USA.
- RHEA, S., GEELS, D., ROSCOE, T. & KUBIATOWICZ, J. (2004). Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, 127–140, Boston, Massachusetts, USA.
- ROWSTRON, A. & DRUSCHEL, P. (2001). Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 329–350, Heidelberg, Germany.
- SCOTT, J., HUI, P., CROWCROFT, J. & DIOT, C. (2006). Hagggle: A networking architecture designed around mobile users. In *Proceedings of the 3rd Annual IFIP Conference on Wireless On-demand Network Systems and Services (WONS 2006)*, Les Ménuires, France.
- SEGALL, B. & ARNOLD, D. (1997). Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, Brisbane, Australia.
- SHEPLER, S., CALLAGHAN, B., ROBINSON, D., THURLOW, R., BEAME, C., EISLER, M. & NOVECK, D. (2003). RFC 3530 - Network file system (NFS) version 4 protocol.
- SINGH, A. & LIU, L. (2003). Trustme: Anonymous management of trust relationships in decentralized p2p. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P03)*, 142–149, Linköping, Sweden.

- SNOEREN, A.C. & BALAKRISHNAN, H. (2000). An end-to-end approach to host mobility. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networks (Mobicom 2000)*, 155–166, Boston, Massachusetts, USA.
- SONG, L., KOTZ, D., JAIN, R. & HE, X. (2006). Evaluating next-cell predictors with extensive wi-fi mobility data. *IEEE Transactions on Mobile Computing*, **5**, 1633–1649.
- SONY COMPUTER ENTERTAINMENT (2007). PlayStation.com - PLAYSTATION®3 - Network - Updates. Retrieved 21 June 2007 from http://www.us.playstation.com/PS3/Network/Updates/PS3_181_update.html#remote_play.
- SPRINT NEXTEL (2007). Sprint music store. Retrieved 9 July 2007 from <http://www1.sprintpcs.com/explore/ueContent.jsp?scTopic=music192>.
- SQUID (2007). Squid : Optimising web delivery. retrieved 27 July 2007 from <http://www.squid-cache.org>.
- STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F. & BALAKRISHNAN, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, 149–160.
- STOICA, I., ADKINS, D., ZHUANG, S., SHENKER, S. & SURANA, S. (2002). Internet indirection infrastructure. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'02)*, 73–86, Pittsburgh, Pennsylvania, USA.
- SUN MICROSYSTEMS (2001). Jini architecture specification version 1.2. Retrieved 20 June 2007 from <http://www.sun.com/software/jini/specs/>.
- SUN MICROSYSTEMS (2002). Java message service specification version 1.1. Retrieved 20 June 2007 from <http://java.sun.com/products/jms/>.
- SURI, N., TORTONESI, M., ARGUEDAS, M., BREEDY, M., CARVALHO, M. & WINKLER, R. (2005). Mockets: a comprehensive application-level communications library. In *Proceedings of the Military Communications Conference (MILCOM 2005)*, 970–976, Atlantic City, New Jersey, USA.
- SUTTON, P., ARKINS, R. & SEGALL, B. (2001). Supporting disconnectedness - transparent information delivery for mobile and invisible computing. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia.
- SYDNEY MORNING HERALD (2007). Sydney WiFi project 'on track'. Retrieved 27 July 2007 from <http://www.smh.com.au/news/tech/sydney/2007/07/18/1184559848818.html>.
- T-MOBILE (2007). Wireless internet access - t-mobile hotspots. Retrieved 27 July 2007 from <http://hotspot.t-mobile.com>.
- TAKACHI, K., OTSUKA, T. & MITOMO, H. (2006). A cost benefit analysis of the nationwide FTTH development in Japan. Retrieved 17 July 2007 from <http://www.soumu.go.jp/iicp/chousakenkyu/seika/pdf/2006-05.pdf>.

- TELSTRA (2007). Telstra super-charges Next G network. Retrieved 27 July 2007 from <http://www.telstra.com.au/abouttelstra/media/release.cfm?ObjectID=39138>.
- THE SALUTATION CONSORTIUM (1999). Salutation architecture specification version 2.1.
- THECLOUD.NET (2007). The Cloud switches on Europes most advanced WiFi network across the City of London. Retrieved 27 July 2007 from <http://www.thecloud.net/content.asp?section=5&content=49&expand=1680>.
- TRUSTEES OF DARTMOUTH COLLEGE (2007). Dartmouth maps. Retrieved 28 June 2007 from <http://www.dartmouth.edu/~maps/>.
- UPNP FORUM (2003). UPnP device architecture 1.0. Retrieved 20 June 2007 from <http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf>.
- VARGA, A. (2001). The OMNeT++ discrete event simulation system. In *Proceedings of the 15th European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic.
- VIXIE, P., THOMSON, S., REKHTER, Y. & BOUND, J. (1997). RFC 2136 - Dynamic updates in the domain name system (DNS update).
- WALDO, J. (1999). The Jini architecture for network-centric computing. *Communications of the ACM*, **42**, 76–82.
- WIGLE (2008a). WiGLE – networks over time. Retrieved 2 April 2008 from <http://wigle.net/gps/gps/main/stats/>.
- WIGLE (2008b). WiGLE – wireless geographic logging engine - plotting WiFi on maps. Retrieved 2 April 2008 from <http://wigle.net/gps/gps/main/>.
- WIRELESS PHILADELPHIA (2007). Wireless Philadelphia - make a digital difference. Retrieved 27 July 2007 from <http://www.wirelessphiladelphia.org>.
- WU, G. & CHIUEH, T.C. (2006). How efficient is BitTorrent? In *Proceedings of the 13th Annual Conference on Multimedia Computing and Networking (MMCN'06)*, San Jose, California, USA.
- YAHOO! (2007a). Flickr: Mobile tools. Retrieved 9 July 2007 from <http://www.flickr.com/tools/mobile/>.
- YAHOO! (2007b). Welcome to Flickr - photo sharing. Retrieved 9 July 2007 from <http://www.flickr.com>.
- YAHOO! (2007c). Yahoo! Retrieved 9 July 2007 from <http://www.yahoo.com>.
- YAHOO! (2007d). ZoneTag photos. Retrieved 9 July 2007 from <http://zonetag.research.yahoo.com>.
- ZHAO, B.Y., KUBIATOWICZ, J.D. & JOSEPH, A.D. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, EECS, UC Berkeley.

- ZHAO, B.Y., HUANG, L., JOSEPH, A.D. & KUBIATOWICZ, J. (2004). Rapid mobility via type indirection. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, 64–74, San Diego, California, USA.
- ZHU, F., MUTKA, M.W. & NI, L.M. (2005). Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, **4**, 81–90.
- ZHUANG, S., LAI, K., STOICA, I., KATZ, R. & SHENKER, S. (2003). Host mobility using an internet indirection infrastructure. In *Proceedings of 1st International Conference on Mobile Systems, Applications, and Services (Mobisys 2003)*, 129–144, San Francisco, California, USA.
- ZIV, J. & LEMPEL, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, **24**, 530–536.